APOGEO

IDEE&STRUMENTI

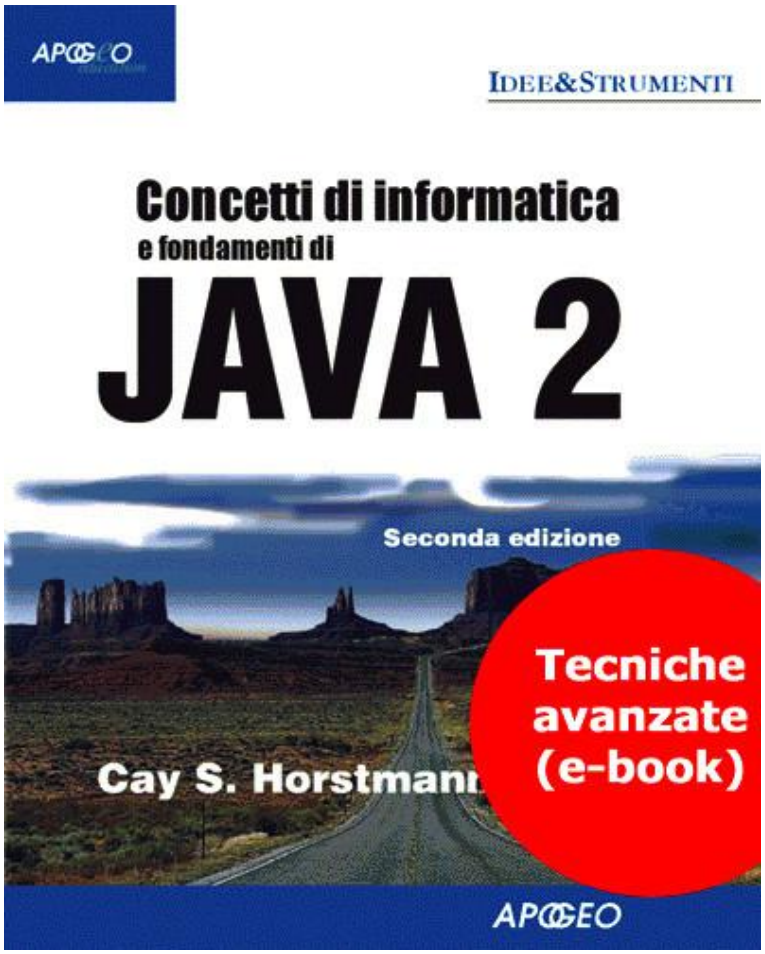
Concetti di informatica
e fondamenti di
JAVA 2

Seconda edizione

Tecniche avanzate (e-book)

Cay S. Horstmann

APOGEO



Concetti di informatica
e fondamenti di
JAVA 2
Seconda edizione
C ay S. Horstmann
Edizione italiana
a cura di Marcello Dalpasso

Concetti di informatica e fondamenti di Java 2
Tecniche avanzate - Edizione elettronica

Titolo originale:
Big Java
Autore:
Cay S. Horstmann

Copyright © **2002 – John Wiley & Sons, Inc. All rights reserved.**

Authorized translation from the English language edition published by John Wiley & Sons, Inc.

Copyright © **2002 – APOGEO**
Via Natale Battaglia 12 – 20127 Milano (Italy)
Telefono: 02-28970277 (5 linee r.a.) – Telefax: 02-26116334
Email
education@apogeonline.com
U.R.L. **http://www.apogeonline.com**
ISBN 88-503-1010-2
Traduzione: Jordanit Ascoli, Marco Meneghini, Michele Pacifico, Marcello Dalpasso Revisione: Marcello Dalpasso
Impaginazione elettronica: Grafica Editoriale – Vimercate Editor: Alberto Kratter Thaler
Copertina e progetto grafico: Enrico Marcandalli Responsabile editoria digitale: Alberto Mari
Tutti i diritti sono riservati a norma di legge e a norma delle convenzioni internazionali. Nessuna parte di questo libro può essere riprodotta con sistemi elettronici, meccanici o altri, senza l'autorizzazione scritta dell'Editore.
Nomi e marchi citati nel testo sono generalmente depositati o registrati dalle rispettive case produttrici.
Multithreading
Obiettivi del capitolo
■ Capire come thread multipli possono essere eseguiti in parallelo
■ Imparare a implementare i thread
■ Capire le condizioni di corsa critica e di blocco fatale
■ Saper evitare i danni a oggetti condivisi usando metodi sincronizzati
■ Saper usare i thread per realizzare animazioni **2**
Multithreading
Spesso è utile che un programma svolga due o più compiti contemporaneamente. Ad esempio, un browser Web può caricare contemporaneamente più immagini contenute in una pagina Web, oppure un programma di animazione può mostrare figure in movimento, con compiti distinti per il calcolo delle posizioni di ciascuna figura. Ovviamente, effetti come questi si possono programmare con un ciclo che prima esegua un po' di lavoro del primo compito, poi un po' del secondo, e così via, ma

tali programmi diventano rapidamente complessi, perché dovete intercalare il codice che esegue il lavoro con il codice che controlla il tempo.

Un thread è un'unità del program-

In questo capitolo vedrete come poter, invece, realizzare ciascun comma che viene eseguita indipen-

pito con un *thread* (flusso di esecuzione), che è un'unità del programma dentemente dalle altre porzioni del

che viene eseguita indipendentemente dalle altre porzioni del program-programma stesso.

ma stesso. La macchina virtuale Java, e non il programmatore, esegue ciascun thread per un breve periodo di tempo, passando poi a un altro thread: ciò fornisce l'illusione che i thread vengano eseguiti in parallelo l'uno all'altro.

A dire il vero, se un computer ha più unità centrali di elaborazione (CPU), allora alcuni thread possono essere veramente eseguiti in parallelo, uno su ciascun processore.

I thread sono comodi per i programmatori: vi consentono di concentrarvi sul compito che dovete portare a termine, senza dovervi preoccupare di quale compito venga alternato a quale altro. Se dovete eseguire due compiti in parallelo, semplicemente realizzate un thread per ciascuno di essi e fateli partire. Ci sono, naturalmente, un paio di complicazioni, perché a volte dovete, in qualche modo, sincronizzare i thread: un thread può aver bisogno di un risultato che viene calcolato da un altro thread. Un altro problema ricorrente si ha quando più thread cercano simultaneamente di modificare un oggetto condiviso. Vedrete in questo capitolo come gestire questi problemi.

1
Nozioni fondamentali dei thread

1.1
Eseguire un thread

Eseguire un thread è semplice, basta seguire questi passi: 1. Implementate una classe che estende la classe Thread.

2. Inserite nel metodo run di tale sottoclasse il codice che esegue il vostro compito.

3. Create un oggetto di tale sottoclasse.

4. Invocate il suo metodo start per far partire il thread.

Quando viene fatto partire un og-

Osserviamo un esempio concreto. Vogliamo soltanto stampare dieci getto di tipo Thread, viene ese-saluti “Hello, World!”, un saluto ogni secondo. Aggiungiamo una in-guito in un nuovo thread il codi-

dicazione dell’orario a ogni saluto, per vedere quando viene stam-ce del suo metodo run.

pato.

Thu Dec 28 23:12:03 PST 2000 Hello, World!
Thu Dec 28 23:12:04 PST 2000 Hello, World!
Thu Dec 28 23:12:05 PST 2000 Hello, World!
Thu Dec 28 23:12:06 PST 2000 Hello, World!
Thu Dec 28 23:12:07 PST 2000 Hello, World!
Thu Dec 28 23:12:08 PST 2000 Hello, World!
Thu Dec 28 23:12:09 PST 2000 Hello, World!

Thu Dec 28 23:12:10 PST 2000 Hello, World!
Thu Dec 28 23:12:11 PST 2000 Hello, World!
Thu Dec 28 23:12:12 PST 2000 Hello, World!

Seguendo le istruzioni per creare un thread, definiamo una classe che estende la classe Thread:

```
public class GreetingThread extends Thread
{
public void run()
{
// azioni del thread
...
}
// variabili usate dalle azioni del thread
...
}
```

Ecco le azioni per i nostri thread:

- Stampare l’indicazione dell’orario.
- Stampare il saluto.
- Attendere un secondo.

Il thread dovrebbe eseguire il ciclo dieci volte.

Per avere un’indicazione dell’orario costruiamo un oggetto di tipo Date, il cui costruttore predefinito fornisce l’ora e la data attuale.

Date now = new Date();

System.out.println(now + “ “ + greeting);

Il metodo sleep pone in uno stato

Per aspettare un secondo, usiamo il metodo sleep della classe Thread.

dormiente il thread attuale per il

L’invocazione

numero di millisecondi specifi-

cato.

sleep(milliseconds)

pone in uno stato dormiente il thread attuale per il numero di millisecondi specificato.

Nel nostro caso vogliamo un’attesa di 1000 millisecondi, cioè un secondo.

Rimane ancora un problema tecnico: porre in uno stato dormiente un thread è un’azione potenzialmente pericolosa, perché potrebbe rimanervi così al ungo da non essere più utile e da rendere necessaria una sua interruzione. Come vedrete nel Paragrafo 1.2, per porre fine a un thread, lo si *interrompe*. Quando un thread dormiente viene interrotto, si genera un’eccezione di tipo InterruptedException, che va cattura-ta nel metodo run, per poi terminare il thread. Il modo più semplice di gestire le interruzioni dei thread è di scrivere il metodo run con lo schema seguente: public void run()

{

try

{

azioni del thread

}

catch (InterruptedException exception)

{

azioni di pulizia, se necessarie

}

}

Quando un thread viene interrotto,

Nel nostro esempio, seguiamo tale struttura; ecco il codice completo la reazione più comune è la ter-della classe del thread.

minazione del metodo run.

File GreetingThread.java

```
import java.util.Date;
/**
Un thread che stampa più volte un saluto.
*/
public class GreetingThread extends Thread
{
/**
Costruisce l’oggetto che rappresenta il thread.
@param aGreeting il saluto da visualizzare
*/
public GreetingThread(String aGreeting)
{
greeting = aGreeting;
}
public void run()
{
try
{
for (int i = 1; i <= REPETITIONS; i++)
{
Date now = new Date();
System.out.println(now + “ “ + greeting);
sleep(DELAY);
}
}
catch (InterruptedException exception)
{
}
}
private String greeting;
private static final int REPETITIONS = 10;
private static final int DELAY = 1000;
```

}

Per far partire un thread, per prima cosa occorre costruire un oggetto della classe che lo rappresenta.

GreetingThread t = new GreetingThread(“Hello, World!”);

Quindi, invocate il metodo start.

t.start();

A questo punto viene fatto partire un nuovo thread, eseguendo il codice del metodo run del vostro thread in parallelo con tutti gli altri thread del programma.

Nel nostro programma di esempio, facciamo partire due thread: uno che stampa

“Hello, World!” e un altro che stampa “Goodbye, World!”.

File GreetingThreadTest.java

```
/**
Questo programma collauda il thread che stampa i saluti, eseguendo due di tali thread in parallelo.
*/
public class GreetingThreadTest
{
public static void main(String[] args)
{
GreetingThread t1 =
new GreetingThread(“Hello, World!”);
GreetingThread t2 =
new GreetingThread(“Goodbye, World!”);
t1.start();
t2.start();
}
}
```

Ecco un esempio di risultato prodotto dall’esecuzione del programma. Poiché i due thread vengono eseguiti in parallelo, i due messaggi appaiono intercalati.

```
Thu Dec 28 23:12:03 PST 2000 Hello, World!
Thu Dec 28 23:12:03 PST 2000 Goodbye, World!
Thu Dec 28 23:12:04 PST 2000 Hello, World!
Thu Dec 28 23:12:05 PST 2000 Hello, World!
```

Thu Dec 28 23:12:04 PST 2000 Goodbye, World!
Thu Dec 28 23:12:05 PST 2000 Goodbye, World!
Thu Dec 28 23:12:06 PST 2000 Hello, World!
Thu Dec 28 23:12:06 PST 2000 Goodbye, World!
Thu Dec 28 23:12:07 PST 2000 Hello, World!
Thu Dec 28 23:12:07 PST 2000 Goodbye, World!
Thu Dec 28 23:12:08 PST 2000 Hello, World!
Thu Dec 28 23:12:08 PST 2000 Goodbye, World!
Thu Dec 28 23:12:09 PST 2000 Hello, World!
Thu Dec 28 23:12:09 PST 2000 Goodbye, World!
Thu Dec 28 23:12:10 PST 2000 Hello, World!
Thu Dec 28 23:12:10 PST 2000 Goodbye, World!
Thu Dec 28 23:12:11 PST 2000 Goodbye, World!
Thu Dec 28 23:12:11 PST 2000 Hello, World!
Thu Dec 28 23:12:12 PST 2000 Goodbye, World!
Thu Dec 28 23:12:12 PST 2000 Hello, World!

6

Multithreading

Il pianificatore (scheduler) dei

Se guardate attentamente, vedrete che i due thread non sono *esatta-thread esegue ciascun thread per*

mente intercalati: a volte, il secondo thread sembra saltare innanzi al un breve intervallo di tempo, detto

primo. Ciò mostra una caratteristica importante dei thread: il pianifica-time slice (finestra temporale).

tore (scheduler) dei thread non fornisce alcuna garanzia sull'ordine in cui vengono i thread. Ogni thread viene eseguito per un breve intervallo di tempo, detto time slice (finestra temporale), dopo di che il pianificatore attiva un altro thread, scegliendolo tra i thread eseguibili a sua disposizione. Un thread è esegui-bile se non è dormiente o bloccato in qualche modo. Tuttavia, i tempi di esecuzione sono sempre un po' variabili, specialmente quando vengono invocati servizi del sistema operativo (come la gestione dell'ingresso e dell'uscita standard), per cui non c'è da meravigliarsi se l'ordine in cui ciascun thread assume il controllo è in qualche modo casuale.

Errori comuni 1

Invocare il metodo** run **di un thread

Quando realizzate la classe di un thread, vi impegnate per realizzare un metodo run che contenga l'azione che deve essere eseguita dal thread stesso. Sembrerebbe, quindi, naturale invocare tale metodo:

MyThread thread = new MyThread();

thread.run(); // NON fa partire un nuovo thread!

Ma il metodo run viene eseguito nel thread attuale, non in un nuovo thread!

Per far partire un nuovo thread, dovete chiamare thread.start(). Il metodo start crea un nuovo thread e, dopo averlo fatto partire, ne invoca il metodo run.

Argomenti avanzati 1

***L'interfaccia** Runnable*

Invece di scrivere l'azione che deve essere eseguita da un thread all'interno del metodo run di una sottoclasse di Thread, potete usare l'interfaccia Runnable: public interface Runnable

{

void run();

}

Definite una classe che implementi l'interfaccia Runnable: public class MyRunnable implements Runnable

{

Multithreading

7

public void run()

{

// azioni del thread

...

}

// altri metodi e variabili

...

}

Quindi, costruite un oggetto di tipo Thread a partire da un esemplare della classe, e fate partire il thread nel solito modo:

MyRunnable runnable = new MyRunnable();

Thread myThread = new Thread(runnable);

myThread.start();

Prima che le classi interne venissero aggiunte al linguaggio Java, questo era un costrutto sintattico molto utile, perché, aggiungendo il metodo run a una classe, si ha accesso alle sue variabili private. Oggi, però, è altrettanto semplice creare la classe del thread come classe interna.

Argomenti avanzati 2

Le priorità dei thread

Potete assegnare diverse priorità ai thread, invocando il metodo setPriority con un parametro che indica la priorità, compresa tra 1 (o Thread.MIN_PRIORITY) e 10 (o Thread.MAX_PRIORITY). Il valore Thread.NORM_PRIORITY è definito uguale a 5. Ad esempio myThread.setPriority(Thread.NORM_PRIORITY + 2);

myThread.start();

Ogni volta che il pianificatore dei thread deve decidere quale thread eseguire, sceglie il thread con la priorità più alta. Se ci sono più thread con la stessa priorità, il pianificatore ne sceglie uno a caso. Ad esempio, se un thread a elevata priorità si risveglia da una chiamata a sleep mentre è in esecuzione un thread a più bassa priorità, il pianificatore sospenderà quest'ultimo per eseguire il primo.

Tuttavia, in pratica le priorità sono piuttosto inaffidabili. Ad esempio, il sistema di gestione dei thread di Windows NT ha soltanto sette livelli di priorità, per cui alcune differenze di priorità non hanno in realtà alcun impatto sul pianificatore.

Le priorità non sono poi così utili per la maggior parte delle attività di programmazione che usano i thread. Se scoprite che state lavorando duramente con i livelli di priorità per far funzionare il vostro programma che sfrutta il multithreading, probabilmente state facendo qualcosa di sbagliato. Nel Paragrafo 2 conoscerete la sincronizza-zione dei thread, uno schema più affidabile per controllare il comportamento dei vostri thread.

1.2

Terminare un thread

Un thread termina quando l'esecuzione del suo metodo run restituisce il controllo al chiamante. Questo è il modo normale per terminare un thread: implementare il metodo run in modo che esegua un enunciato return quando si accorge che non c'è più lavoro da eseguire.

Un thread termina quando termi-

Tuttavia, a volte avete bisogno di porre termine a un thread che è in na il suo metodo run.

esecuzione. Ad esempio, potete avere diversi thread che stanno cercando di trovare una soluzione a un problema. Appena il primo ha successo, è probabile che vogliate interrompere gli altri. Nelle prime versioni della libreria Java, la classe Thread aveva un metodo stop per terminare l'esecuzione di un thread, ma tale metodo è ora sconsigliato (deprecated), perché i teorici dell'informatica hanno scoperto che terminare un thread può portare il sistema in una condizione pericolosa quando più thread condividono alcuni oggetti. (Nel Paragrafo 2 discuteremo l'accesso a oggetti condivisi.) Invece di terminare semplicemente un thread, dovreste segnalare al thread di terminare la propria esecuzione. C'è quindi bisogno della cooperazione del thread, che deve abbandonare tutte le risorse che sta utilizzando ed effettuare altre necessarie operazioni di pulizia.

Per segnalare a un thread che deve terminare la propria esecuzione, usate il metodo interrupt.

t.interrupt():

Un thread dovrebbe, ogni tanto,

Il metodo run di un thread dovrebbe controllare, ogni tanto, se è stato controllare se è stato interrotto,

interrotto, invocando il metodo isInterrupted. In tal caso, dovrebbe chiamando il metodo isInter-compiere le necessarie azioni di pulizia e terminare la propria esecuzio-rupted.

ne. Ad esempio, il metodo run della classe GreetingThread potrebbe fare questo controllo all'inizio di ciascuna iterazione del ciclo: public void run()

```
{
for (int i = 1;
i <= REPETITIONS && !isInterrupted(); i++)
```

{
fa qualcosa
}
fa pulizia
}

Tuttavia, se un thread è dormiente, non può eseguire codice che verifichi se è stato interrotto. Per questo motivo, il metodo sleep lancia una InterruptedException quando viene interrotto un thread dormiente. Dovete, quindi, catturare tale eccezione e terminare il thread se essa viene lanciata. Il modo più semplice di farlo consiste nel circondare l'intera porzione del metodo run che svolge le azioni del thread con un blocco try, in questo modo:

public void run()
{
try
Multithreading
9
{
for (int i = 1;
i <= REPETITIONS && !isInterrupted(); i++)
{
fa qualcosa
}
}
catch (InterruptedException exception)
{
}
fa pulizia
}

A rigore, la specifica del linguaggio Java non dice che un thread debba terminare quando viene interrotto: ciò che fa quando viene interrotto è sotto la sua completa responsabilità. L'interruzione è un metodo generico per richiedere l'attenzione di un thread, anche quando è dormiente. Tuttavia, in questo capitolo termineremo sempre un thread che viene interrotto.

Suggerimenti per la qualità 1

Controllate all'interno del metodo *run*

se un thread viene interrotto

Per convenzione, quando un thread viene interrotto dovrebbe terminare la propria esecuzione (o, perlomeno, compiere qualche azione ben precisa). Dovreste implementare i vostri thread seguendo questa convenzione.

Fate semplicemente così:

- *In un ciclo del vostro thread, invocate il metodo isInterrupted. Se restituisce true, eseguite le necessarie azioni di pulizia e terminate il metodo run.*
- *Inserite le azioni del thread in un blocco try che catturi l'eccezione InterruptedException, che viene lanciata se il thread viene interrotto mentre non è in esecuzione, ad esempio perché ha invocato sleep. Quando catturate l'eccezione, eseguite le necessarie azioni di pulizia e terminate il metodo run.*

Molti programmatori non capiscono lo scopo dell'eccezione InterruptedException e, per ignoranza e disperazione, la zittiscono circondando con un blocco try soltanto l'invocazione del metodo sleep.

```
public void run()
{
while (...)
{
...
try
{
```



```
sleep(tempo);
}
catch (InterruptedException e) {} // NON FATELO
...
}
}
```

Non fate in questo modo. Se fate così, gli utilizzatori della vostra classe thread non potranno ottenere l'attenzione del thread interrompendolo. Inserire l'intera azione del thread in un singolo blocco try è altrettanto semplice, ma almeno, in tal modo, interrompendo il thread si ottiene la sua terminazione.

```
public void run()
{
try
{
while (...)
{
...
sleep(tempo);
...
}
}
catch (InterruptedException e) {} // Va bene
}
```

Argomenti avanzati 3

I gruppi di thread

Un programma con molti thread può inserirli in gruppi di thread, che si creano con il seguente costruttore:

```
String name = "...";
ThreadGroup group = new ThreadGroup(name);
```

Successivamente, aggiungete i thread al gruppo, fornendo il gruppo nel loro costruttore, come in questo esempio:

```
Thread myThread = new Thread(group, myRunnable);
```

A questo punto, anche qualsiasi thread che venga fatto partire da myThread apparterrà automaticamente al nuovo gruppo di thread.

Il principale vantaggio dei gruppi di thread è che tutti i thread di un gruppo possono essere gestiti unitariamente. Ad esempio, per interrompere tutti i thread di un gruppo, invocate

```
group.interrupt();
```


Come esempio di applicazione pratica, potete immaginare che tutti i thread che servono a caricare le varie parti di una pagina Web vengano inseriti in un gruppo: se a un certo punto l'utente decide che la pagina non venga più caricata, potete semplicemente interrompere tutti i thread con un solo enunciato.

Sincronizzazione

2.1

Problemi causati da thread non sincronizzati

Quando diversi thread condividono l'accesso a un oggetto comune, possono entrare in conflitto l'uno con l'altro. Per mettere in evidenza i problemi che possono sorgere, analizzeremo un programma in cui più thread effettuano operazioni su un conto bancario. Ciascun thread versa o preleva ripetutamente una certa somma di denaro, quindi entra nello stato dormiente per un breve periodo di tempo. Ecco il metodo run della classe DepositThread:

```
public void run()
{
try
{
for (int i = 1;
i <= REPETITIONS && !isInterrupted(); i++)
{
account.deposit(amount);
sleep(DELAY);
}
}
catch (InterruptedException exception)
{
}
}
```

La classe WithdrawThread è simile, soltanto che preleva denaro.

Nel nostro programma di esempio, costruiamo un conto bancario con un saldo iniziale nullo, poi creiamo due thread:

- *il thread t0 versa $ 100 nel conto bancario, eseguendo 10 iterazioni.*
- *il thread t1 preleva $ 100 dal conto bancario, eseguendo 10 iterazioni.*

I metodi deposit e withdraw della classe BankAccount sono stati modificati per visualizzare messaggi che mostrano ciò che accade. Ad esempio, ecco il codice del metodo deposit:

public void deposit(double amount)
{
System.out.print("Depositing " + amount);
12
Multithreading
double newBalance = balance + amount;
System.out.println(", new balance is " + newBalance); balance = newBalance;
}
Al termine del paragrafo troverete il codice sorgente completo.
La visualizzazione prodotta dal programma assomiglierà, solitamente, a questa: Depositing 100.0, new balance is 100.0
Withdrawing 100.0, new balance is 0.0
Depositing 100.0, new balance is 100.0
Depositing 100.0, new balance is 200.0
Withdrawing 100.0, new balance is 100.0
...
Withdrawing 100.0, new balance is 0.0
Alla fine il saldo dovrebbe essere zero, ma, eseguendo il programma ripetutamente, potrete a volte notare una visualizzazione confusa, come questa: Depositing 100.0Withdrawing 100.0, new balance is –100.0
, new balance is 100.0
Se osservate l'ultima riga della visualizzazione, noterete che il saldo finale non è sempre zero. Evidentemente, è successo qualcosa di strano.
Per osservare questo effetto, può darsi che dobbiate eseguire il programma più volte.
Ecco uno scenario che spiega come possa accadere questo problema.
1. Il primo thread esegue le righe
System.out.print("Depositing " + amount);
double newBalance = balance + amount;
nel metodo deposit della classe BankAccount. Il valore del campo balance è ancora 0 e il valore della variabile locale newBalance è 100.

2. Immediatamente dopo, il primo thread raggiunge la fine del suo intervallo temporale e il controllo viene assunto dal secondo thread.

3. Il secondo thread invoca il metodo withdraw, che visualizza il proprio messaggio e preleva $100 dalla variabile balance, che vale ora –100.

4. Il secondo thread diventa dormiente.

5. Il primo thread riprende il controllo e riparte dal punto in cui si era interrotto, eseguendo le linee

System.out.println(“, new balance is “ + newBalance); balance = newBalance;

Il valore di balance è ora 100 (osservate la Figura 1).

In questo modo, non solo i messaggi sono intercalati erroneamente, ma anche il saldo è sbagliato. Il saldo dopo un prelievo e un versamento dovrebbe essere 0, non 100.

Poiché il metodo deposit è stato interrotto, ha usato il vecchio saldo (prima del prelie-Multithreading

Figura 1

Corruzione
del contenuto
del campo balance

vo) per calcolare il valore della propria variabile locale newBalance; poi, quando ha ripreso l’esecuzione, ha usato il valore di newBalance per sovrascrivere il campo balance, che nel frattempo era stato modificato.

Si ha una condizione di corsa

Come potete vedere, ciascun thread ha le sue variabili locali, ma en-critica quando l’effetto comples-

trambi condividono l’accesso alla variabile istanza balance: tale accesso sivo di più thread che condivi-condiviso è la fonte del problema, che viene spesso chiamato condizione dono dati dipende dall’ordine in

di corsa critica (race condition). Entrambi i thread, nella loro corsa verso il cui i thread vengono eseguiti.

completamento dei rispettivi compiti, manipolano un campo condiviso, e il risultato finale dipende da quale thread vince la corsa.

Potreste pensare che l'origine di questo problema stia nel fatto che abbiamo reso troppo semplice la possibilità di interrompere il calcolo del saldo. Supponete che il codice del metodo deposit venga riorganizzato in questo modo: public void deposit(double amount)

```
{
balance = balance + amount;
System.out.println("Depositing " + amount
+ ", new balance is " + balance);
}
```

Supponete, inoltre, di fare la stessa modifica nel metodo withdraw. Se eseguite il programma che ne deriva, tutto sembra andar bene.

Tuttavia, questa è una pericolosa illusione. Il problema non è stato risolto, è soltanto divenuto meno frequente, e quindi più difficile da osservare. È ancora possibile che il metodo deposit raggiunga la fine del suo intervallo temporale dopo aver calcolato il valore che si trova alla destra dell'operazione di assegnamento balance + amount

ma prima di eseguire l'assegnamento stesso

balance = il valore del membro destro dell'operatore Quando il metodo riprende il controllo dell'esecuzione, porta finalmente a termine l'assegnamento, inserendo il valore errato nel campo balance.

File BankAccountThreadTest.java

```
/**
Questo programma esegue due thread che versano e
prelevano denaro dallo stesso conto bancario.
*/
public class BankAccountThreadTest
{
public static void main(String[] args)
{
BankAccount account = new BankAccount();
DepositThread t0 =
```

```
new DepositThread(account, 100);
WithdrawThread t1 =
new WithdrawThread(account, 100);
```

```
t0.start();
t1.start();
}
}
```
File DepositThread.java
```
/**
Un thread di versamento esegue versamenti periodici in un conto bancario.
*/
class DepositThread extends Thread
{
/**
Costruisce un thread di versamento.
@param anAccount il conto in cui versare denaro
@param anAmount la somma da versare ogni volta
*/
public DepositThread(BankAccount anAccount,
double anAmount)
{
account = anAccount;
amount = anAmount;
}
public void run()
{
try
{
for (int i = 1;
i <= REPETITIONS && !isInterrupted(); i++)
{
account.deposit(amount);
sleep(DELAY);
}
}
```

```
catch (InterruptedException exception)
{
}
}
private BankAccount account;
private double amount;
private static final int REPETITIONS = 10;
private static final int DELAY = 10;
}
```

File WithdrawThread.java

```
/**
Un thread di prelievo esegue prelievi periodici da un conto bancario.
*/
```



```
class WithdrawThread extends Thread
{
/**
Costruisce un thread di prelievo.
@param anAccount il conto da cui prelevare denaro
@param anAmount la somma da prelevare ogni volta
*/
public WithdrawThread(BankAccount anAccount,
double anAmount)
{
account = anAccount;
amount = anAmount;
}
public void run()
{
try
{
for (int i = 1;
i <= REPETITIONS && !isInterrupted(); i++)
{
account.withdraw(amount);
sleep(DELAY);
```

```
}
}
catch (InterruptedException exception)
{
}
}
private BankAccount account;
private double amount;
private static final int REPETITIONS = 10;
private static final int DELAY = 10;
}
```

File BankAccount.java

```
/**
Un conto bancario ha un saldo che può essere modificato con versamenti e prelievi.
*/
public class BankAccount
{
/**
Costruisce un conto bancario con saldo zero.
*/
public BankAccount()
{
balance = 0;
}
```



```
/**
Versa denaro nel conto bancario.
@param amount la somma da versare
*/
public void deposit(double amount)
{
System.out.print("Depositing " + amount);
double newBalance = balance + amount;
System.out.println(", new balance is " + newBalance); balance = newBalance;
}
```

```
/**
Preleva denaro dal conto bancario.
@param amount la somma da prelevare
*/
public void withdraw(double amount)
{
System.out.print("Withdrawing " + amount);
double newBalance = balance - amount;
System.out.println(", new balance is " + newBalance);
balance = newBalance;
}
/**
Fornisce il saldo attuale del conto bancario.
@return il saldo attuale
*/
public double getBalance()
{
return balance;
}
private double balance;
}
```

2.2 Sincronizzare l'accesso agli oggetti

Per risolvere problemi come quello che avete visto nel paragrafo precedente, un thread deve essere in grado di bloccare (lock) temporaneamente un oggetto. Mentre il thread ha bloccato un oggetto, nessun altro thread dovrebbe poter modificare lo stato dell'oggetto.

Nel linguaggio di programmazione Java, si usano i metodi sincronizzati per bloccare un oggetto. Dovete contrassegnare con la parola chiave synchronized tutti i metodi che contengono codice sensibile all'esecuzione in thread multipli, come i metodi deposit e withdraw della classe BankAccount.

```
public class BankAccount
{
public synchronized void deposit(double amount)
{
```

Multithreading

...

}

public synchronized void withdraw(double amount)

{

...

}

...

}

Eseguendo un metodo sincroniz-

Ciascun oggetto può essere bloccato e, per impostazione predefinita, zato, un thread diventa titolare del

non è bloccato. Un thread lo può bloccare eseguendo un metodo sin-blocco dell'oggetto, dopo di che

cronizzato: a quel punto, quel thread è titolare del blocco finché il metodo nessun altro thread può blocca-termina, sbloccando così l'oggetto. Quando un oggetto è bloccato da re lo stesso oggetto.

un thread, nessun altro thread può iniziare l'esecuzione di un metodo sincronizzato di tale oggetto. Se un altro thread invoca un metodo sincronizzato dell'oggetto bloccato, perde automaticamente il controllo dell'esecuzione e deve attendere finché il thread che è titolare del blocco revoca il blocco stesso.

Un modo per visualizzare questo comportamento è immaginare che l'oggetto sia un bagno pubblico e che i thread siano persone. Il bagno può accogliere una sola persona, che ne usa i servizi. Se il bagno è vuoto, allora la prima persona che lo vuole usare vi entra e chiude la serratura della porta (lo blocca). Se un'altra persona vuole usare il bagno e lo trova bloccato, deve aspettare finché la prima persona esce dal bagno. Se più persone vogliono entrare in bagno, aspetteranno tutte fuori: non è necessario che formino una coda ordinata, è anche possibile che una persona scelta a caso possa entrare quando il bagno diventa libero.

Dichiarando che entrambi i metodi, deposit e withdraw, sono sincronizzati, ci as-sicuriamo che il nostro programma venga sempre eseguito correttamente: un solo thread per volta può eseguire uno dei due metodi su un certo oggetto. Ogni volta

che un metodo inizia l'esecuzione di uno di tali metodi, ha la certezza di eseguirlo fino al termine prima che un altro thread abbia la possibilità di eseguire un metodo sincronizzato sullo stesso oggetto.

Figura 2

Rappresentazione

del blocco

di un oggetto

Multithreading

19

3

Evitare blocchi fatali

Un blocco fatale (deadlock) ac-

Potete usare il blocco di oggetti e i metodi sincronizzati per essere certi cade se nessun thread può pro-che i vostri oggetti si trovino in uno stato consistente quando diversi cedere nell'esecuzione perché

thread vi accedono. Tuttavia, queste tecniche posso anche dare origine a ciascun thread è in attesa che un

altro thread bloccato produca pri-

un altro problema. Può succedere che un thread blocchi un oggetto ma ma qualche risultato.

poi si metta in attesa che un altro thread produca un risultato essenziale. Se tale altro thread è in attesa di poter bloccare lo stesso oggetto, nessuno dei due thread potrà procedere nell'esecuzione e il programma risulterà bloccato. Una simile situazione viene detta blocco fatale (deadlock). Vediamone un esempio.

Supponete di voler impedire saldi negativi per i conti bancari del vostro programma. Ecco un modo molto semplice di farlo: nel metodo run della classe WithdrawThread, possiamo controllare il saldo prima di prelevare denaro.

```
if (account.getBalance() >= amount)
account.withdraw(amount);
```

Questo funziona se c'è un solo thread in esecuzione che cerca di prelevare denaro, ma supponiamo di avere più thread che prelevano denaro. In questo caso, l'intervallo temporale del

thread attuale può terminare dopo che la verifica account.getBalance()

>= amount è stata verificata con successo, ma prima che il metodo withdraw venga invocato. Se nel frattempo un altro thread preleva ulteriore denaro, la verifica appena eseguita era inutile, e potremmo comunque avere un saldo negativo.

Ovviamente, la verifica dovrebbe essere inserita nel metodo sincronizzato withdraw: in questo modo, la verifica della presenza di fondi sufficienti e il reale prelievo non possono avvenire separatamente. Il metodo withdraw, quindi, dovrebbe assomigliare a questo:

```
public synchronized void withdraw(double amount)
{
while (balance < amount)
aspetta che il saldo aumenti
...
}
```

Come possiamo aspettare che il saldo aumenti? Non potete semplicemente chiamare sleep dal metodo withdraw, perché se un thread entra nello stato dormiente all'interno di un metodo sincronizzato, non revoca il blocco di cui è titolare, per cui nessun altro thread potrebbe eseguire il metodo sincronizzato deposit. Altri thread possono, in verità, invocare il metodo deposit, ma rimarrebbero semplicemente bloccati finché il metodo withdraw non termina. Ma il metodo withdraw non terminerà finché non ci saranno fondi disponibili: ecco la situazione di blocco fatale a cui avevamo accennato in precedenza.

Questa situazione è abbastanza comune ed esiste uno speciale insieme di metodi che sono stati progettati per venire in aiuto in questi casi. Il metodo wait revoca temporaneamente il blocco acquisito su un oggetto e rende inattivo il thread attualmente in esecuzione.

L'invocazione di wait all'interno

Proseguendo con l'analogia del bagno pubblico, non volete che la di un metodo sincronizzato met-persona chiusa nel

bagno si addormenti se è terminata la carta igienica: te in attesa il thread attuale e

è meglio che la persona all'interno rinunci a usare i servizi ed esca. In consente a un altro thread di bloc-questo modo un'altra persona ha la possibilità di entrare nel bagno e di care l'oggetto.

rifornirlo di carta igienica.

Ecco il metodo withdraw corretto.

public synchronized void withdraw(double amount) throws InterruptedException

{

while (balance < amount)

wait();

...

}

Il metodo wait può lanciare un'eccezione di tipo InterruptedException. Il metodo withdraw propaga tale eccezione, perché non ha modo di sapere come vuole reagire all'interruzione il thread che ha invocato il metodo withdraw.

Un thread che ha invocato wait

Quando un thread invoca wait, non viene semplicemente reso inat-rimane bloccato finché un altro

tivo allo stesso modo di un thread che giunge al termine del proprio thread invoca notifyAll o

intervallo temporale, ma viene posto in uno stato bloccato e non verrà notify sull'oggetto per il quale

attivato dal pianificatore dei thread finché non sarà sbloccato. Per sbloc-il thread sta aspettando.

carlo, un altro thread deve eseguire il metodo notifyAll. Il metodo notifyAll sblocca tutti i thread che erano in attesa su quell'oggetto: da questo momento in poi possono tornare a competere con tutti gli altri thread che siano in attesa di acquisire il blocco dell'oggetto.

Nell'analogia del bagno pubblico, il thread che invoca wait corrisponde alla persona che è entrata nel bagno e si è accorta che la carta igienica è terminata. Tale persona, quindi, esce dal bagno e aspetta fuori, senza fare assolutamente nulla, anche se altre persone entrano ed escono dal bagno, perché sa che non

ha senso riprovare. A un certo punto, un inserviente entra nel bagno, lo rifornisce di carta igienica e ne da comunicazione a voce alta. A questo punto tutte le persone in attesa della carta igienica escono dallo stato di inattività e si danno nuovamente da fare per entrare in bagno.

Dovete invocare il metodo notifyAll ogni volta che lo stato di un oggetto è stato modificato in un modo che possa essere utile ad altri thread in attesa. Nel nostro esempio del conto bancario, ciò avviene dopo aver eseguito un versamento. A quel punto, i thread in attesa di un saldo più elevato dovrebbero venire sbloccati, in modo da controllare di nuovo il saldo. Ecco come dovreste modificare il metodo deposit:

```
public synchronized void deposit(double amount)
{
...
notifyAll(); // sblocca tutti i thread in attesa
}
```

Il metodo notifyAll può essere invocato soltanto da un thread che ha bloccato l'oggetto; in altre parole, l'invocazione di notifyAll deve essere all'interno di un metodo Multithreading

sincronizzato o di un metodo che viene chiamato direttamente o indirettamente da un metodo sincronizzato.

Esiste anche il metodo notify, che sceglie a caso un solo thread che sta aspettando per l'oggetto e lo sblocca. Il metodo notify può essere più efficiente, ma è utile soltanto se sapete che tutti i thread in attesa possono procedere effettivamente nella propria esecuzione. In generale questo non si sa e notify può condurre a blocchi fatali. Per questa ragione, consigliamo di usare sempre notifyAll.

Notate che i metodi wait, notifyAll e notify appartengono alla classe Object.

Ciascun oggetto ha una propria coda di thread in attesa e, se chiamate x.wait(), il thread attualmente in esecuzione viene aggiunto alla coda dell'oggetto x. Più comunemente chiamerete wait(), che aggiunge il thread attualmente in esecuzione alla coda dell'oggetto this. In modo simile, l'invocazione di

notifyAll() sblocca tutti i thread che sono in attesa di poter bloccare this.

Usando le invocazioni di wait e notifyAll nei metodi deposit e withdraw, possiamo iniziare un numero qualsiasi di thread di prelievo e versamento senza rischiare un blocco fatale. Se eseguite il programma di esempio, noterete che tutte le transazioni vengono eseguite senza che si raggiunga mai un saldo negativo. Ecco un programma di esempio con quattro thread: due per prelievi e due per versamenti.

File BankAccountThreadTest.java

```
/**
Questo programma esegue quattro thread che versano e prelevano denaro dallo stesso conto bancario.
*/
public class BankAccountThreadTest
{
public static void main(String[] args)
{
BankAccount account = new BankAccount();
DepositThread t0 =
new DepositThread(account, 100);
WithdrawThread t1 =
new WithdrawThread(account, 100);
DepositThread t2 =
new DepositThread(account, 100);
WithdrawThread t3 =
new WithdrawThread(account, 100);
t0.start();
t1.start();
t2.start();
t3.start();
}
}
```

File BankAccount.java

```
/**
Un conto bancario ha un saldo che può essere modificato con versamenti e prelievi.
*/
```

```
public class BankAccount
{
/**
Costruisce un conto bancario con saldo zero.
*/
public BankAccount()
{
balance = 0;
}
/**
Versa denaro nel conto bancario.
@param amount la somma da versare
*/
public synchronized void deposit(double amount)
{
System.out.print("Depositing " + amount);
double newBalance = balance + amount;
System.out.println(", new balance is " + newBalance);
balance = newBalance;
notifyAll();
}
/**
Preleva denaro dal conto bancario.
@param amount la somma da prelevare
*/
public synchronized void withdraw(double amount) throws
InterruptedException
{
while (balance < amount)
wait();
System.out.print("Withdrawing " + amount);
double newBalance = balance - amount;
System.out.println(", new balance is " + newBalance);
balance = newBalance;
}
/**
```

Fornisce il saldo attuale del conto bancario.
@return il saldo attuale
**/*
public double getBalance()
{
return balance;
}
private double balance;
}

Errori comuni 2

***Chiamare** wait **senza chiamare** notifyAll Spesso è facile intuire quando sia il caso di chiamare wait: se un thread scopre di non essere in grado di portare a termine il proprio compito, deve aspettare. Ma, una volta che un thread ha chiamato wait, rinuncia temporaneamente a qualsiasi speranza e non fa ulteriori tentativi finché un altro thread non invoca notifyAll sullo stesso oggetto per il quale il thread sta aspettando. Nell'analogia del bagno pubblico, se l'inserviente che porta la nuova carta igienica non avverte la gente in attesa, essi aspetteranno per sempre.*

Avere dei thread che invocano wait senza le corrispondenti invocazioni di notifyAll in altri thread è un errore comune. Ogni volta che chiamate wait, chiedetevi quale sia l'invocazione di notifyAll che avviserà il vostro thread.

Errori comuni 3

Chiamare** notifyAll **senza aver bloccato l'oggetto

Il thread che chiama notifyAll deve essere titolare del blocco dell'oggetto per il quale invoca notifyAll, altrimenti viene lanciata l'eccezione IllegalMonitorStateException.

Nell'analogia del bagno pubblico, l'inserviente deve annunciare la nuova situazione mentre si trova all'interno del bagno, dopo averlo rifornito di carta igienica.

In pratica, questo non dovrebbe essere un problema. Ricordate che notifyAll viene chiamato quando un thread ha appena modificato lo stato di un oggetto in modo che altri thread ne possano trarre beneficio. Tale cambiamento dovrebbe

trovarsi al-l'interno di un metodo sincronizzato: se contrassegnate come synchronized tutti i metodi che modificano un oggetto condiviso e acquisite l'abitudine di chiamare notifyAll dopo ogni modifica efficace, non avrete problemi. Ma se usate notifyAll a caso, può darsi che vi troviate di fronte a un'eccezione di tipo IllegalMonitorStateException.

Argomenti avanzati 4

Situazioni complesse di blocco fatale

L'utilizzo dei metodi wait e notify non risolve tutti i blocchi fatali: ecco un esempio.

Abbiamo tre conti bancari: account0, account1 e account2, e abbiamo tre thread di trasferimento.

24

Multithreading

Un thread di trasferimento trasferisce denaro da due conti a un terzo conto.

- *t0 trasferisce ripetutamente $ 500 da ciascuno dei conti account1 e account2 verso il conto account0.*
- *t1 trasferisce ripetutamente $ 500 da ciascuno dei conti account2 e account0 verso il conto account1.*
- *t2 trasferisce ripetutamente $ 500 da ciascuno dei conti account1 e account0 verso il conto account2.*

Se i tre thread vengono eseguiti a turno e ciascun thread esegue un'iterazione prima di perdere il controllo dell'esecuzione, allora il programma continua indefinitamente la sua esecuzione:

account0 account1 account2

Saldo iniziale

$ 1000

$ 1000

$ 1000

Dopo l'iterazione di t0

$ 2000

$ 500

$ 500

Dopo l'iterazione di t1

$ 1500

$ 1500

0
Dopo l'iterazione di t2
$ 1000
$ 1000
$ 1000

Ma se accade che t0 esegua due iterazioni prima che t1 e t2 abbiano la possibilità di assumere il controllo, allora il programma entra in un blocco fatale: account0 account1 account2

Saldo iniziale
$ 1000
$ 1000
$ 1000
Dopo l'iterazione di t0
$ 2000
$ 500
$ 500
Dopo l'iterazione di t0
$ 3000
0
0

Ora nessuno dei tre thread può procedere e il programma rimane bloccato fatalmen-te. Potete provare a eseguire il programma: la maggior parte delle volte porterà a termine i trasferimenti con successo, ma una volta ogni tanto potrete osservare un blocco fatale.

Questo blocco fatale può essere evitato riordinando i prelievi, ma non è noto alcun metodo generale per evitare in modo automatico tutti i blocchi fatali. In più, come avete già visto, eseguire soltanto pochi esperimenti può essere molto fuorviante. I programmatori che progettano programmi che utilizzano la tecnica del multithreading devono fare uno sforzo per dimostrare in modo rigoroso che i loro thread non possono provocare un blocco fatale, indipendentemente dall'ordine in cui vengono eseguiti.

File BankAccountThreadTest.java

*/**

Questo programma esegue tre thread che trasferiscono denaro fra tre conti bancari. A volte entra in uno stato di blocco fatale.

```
*/
public class BankAccountThreadTest
{
public static void main(String[] args)
{
```

Multithreading

25

```
BankAccount account0 = new BankAccount(1000);
BankAccount account1 = new BankAccount(1000);
BankAccount account2 = new BankAccount(1000);
TransferThread t0 = new TransferThread(
account1, account2, account0, 500);
TransferThread t1 = new TransferThread(
account2, account0, account1, 500);
TransferThread t2 = new TransferThread(
account1, account0, account2, 500);
t0.start();
t1.start();
t2.start();
}
}
```

File TransferThread.java

```
/**
```

Un thread di trasferimento trasferisce ripetutamente denaro fra tre conti bancari.

```
*/
class TransferThread extends Thread
{
/**
```

Costruisce un thread di trasferimento.

@param account1 il primo conto da cui prelevare

@param account2 il secondo conto da cui prelevare

@param account3 il conto in cui versare

@param anAmount la somma da prelevare da ciascuno dei primi due conti

```
*/
public TransferThread(BankAccount account1,
BankAccount account2, BankAccount account3,
double anAmount)
{
from1 = account1;
from2 = account2;
to = account3;
amount = anAmount;
}
public void run()
{
try
{
for (int i = 1;
i <= REPETITIONS && !isInterrupted(); i++)
{
from1.withdraw(amount);
from2.withdraw(amount);
to.deposit(2 * amount);
```



```
sleep(DELAY);
}
}
catch (InterruptedException exception)
{
}
}
private BankAccount from1;
private BankAccount from2;
private BankAccount to;
private double amount;
private static final int REPETITIONS = 10;
private static final int DELAY = 10;
}
```

File BankAccount.java

```
/**
```

Un conto bancario ha un saldo che può essere modificato con versamenti e prelievi.

```
*/
public class BankAccount
{
/**
Costruisce un conto bancario con un saldo iniziale assegnato.
@param initialBalance il saldo iniziale
*/
public BankAccount(double initialBalance)
{
balance = initialBalance;
}
/**
Versa denaro nel conto bancario.
@param amount la somma da versare
*/
public synchronized void deposit(double amount)
{
balance = balance + amount;
System.out.println(Thread.currentThread().getName()
+ " Depositing " + amount
+ ", new balance is " + balance);
notifyAll();
}
/**
Preleva denaro dal conto bancario.
@param amount la somma da prelevare
*/
public synchronized void withdraw(double amount) throws
InterruptedException
{
```



```
while (balance < amount)
{
System.out.println(Thread.currentThread().getName()
+ " Waiting…");
```

```
wait();
}
balance = balance - amount;
System.out.println(Thread.currentThread().getName()
+ " Withdrawing " + amount
+ ", new balance is " + balance);
}
/**
Fornisce il saldo attuale del conto bancario.
@return il saldo attuale
*/
public double getBalance()
{
return balance;
}
private double balance;
}
```

4
Un'applicazione dei thread: Algoritmi animati

Un utilizzo frequente della programmazione con i thread consiste nella realizzazione di animazioni. Un programma che presenta un'animazione mostra diversi oggetti che si spostano o si modificano in qualche modo con il trascorrere del tempo. Questo effetto viene spesso raggiunto eseguendo uno o più thread che calcolano come si modificano le varie parti dell'animazione.

Per animazioni semplici, potete usare la classe Timer del pacchetto Swing, senza dover realizzare alcuna programmazione di thread: guardate l'Esercizio P8 per un esempio. Tuttavia, animazioni più avanzate si realizzano meglio con i thread.

In questo paragrafo, vedrete un tipo particolare di animazione: la visualizzazione dei passi di esecuzione di un algoritmo. L'animazione degli algoritmi è una tecnica eccellente per raggiungere una migliore comprensione di come funzionino gli algoritmi stessi. Seguendo la tradizione, realizzeremo l'animazione come applet, dato che mettere a disposizione

animazioni su una pagina Web sembra essere una delle attività preferite dai programmatori (questo, come tutti gli applet di questo libro, può essere visualizzato soltanto con un browser abilitato a Java 2, come Netscape 6 o Opera).

Molti algoritmi possono essere animati: provate a scrivere "Java algorithm animation"

nel vostro motore di ricerca per il Web preferito, e troverete un sacco di collegamenti a pagine che contengono animazioni dei più svariati algoritmi.

Tutte le animazioni di algoritmi hanno la stessa struttura. L'algoritmo viene eseguito in un thread apposito, che periodicamente aggiorna un'immagine che rappre-28

Multithreading

senta lo stato attuale dell'algoritmo, per poi fare una pausa e consentire all'utente di osservare tale immagine. Dopo un breve periodo di tempo, il thread di esecuzione dell'algoritmo esce dallo stato dormiente e procede fino al successivo punto interessante dell'algoritmo, aggiornando di nuovo l'immagine e facendo una nuova pausa.

Questa sequenza di eventi viene ripetuta finché l'algoritmo giunge al termine della propria esecuzione.

Prendiamo come esempio l'algoritmo di ordinamento per selezione, che ordina un array di valori. Per prima cosa cerca l'elemento minore, esaminando tutti gli elementi dell'array, e lo porta nella posizione più a sinistra. Quindi, cerca l'elemento minore tra gli elementi rimanenti e lo porta nella seconda posizione. Procedendo in questo modo, la parte ordinata dell'array cresce mentre l'algoritmo viene eseguito.

Come si può visualizzare il funzionamento di questo algoritmo? Ad esempio, è utile evidenziare con un colore diverso la porzione dell'array che è già ordinata. Ancora, vogliamo mostrare come ciascun passo dell'algoritmo esamini un nuovo elemento nella porzione non ordinata. Questo fatto evidenzia perché l'algoritmo di ordinamento per selezione sia tanto lento: per prima cosa esamina tutti gli elementi dell'array, poi tutti tranne uno, e così via. Se l'array ha n elementi, l'algoritmo esamina un numero di elementi uguale a n + (n – 1)

+ (n – 2) + … = n(n – 1)/2 = O(n 2). Per dimostrare ciò, indichiamo in rosso l'elemento attualmente visitato.

Quindi, lo stato dell'algoritmo è descritto da tre entità:

- *i valori contenuti nell'array*
- *la dimensione della porzione già ordinata*
- *l'elemento attualmente visitato*

Dapprima, dovete aggiungere una variabile istanza applet alla classe dell'algoritmo, modificandone il costruttore per impostarla. Tale variabile istanza è necessaria per ridisegnare l'applet e per conoscere le dimensioni dell'applet quando si traccia graficamente lo stato dell'algoritmo.

```
public class SelectionSorter
{
public SelectionSorter(int[] anArray, Applet anApplet)
{
a = anArray;
applet = anApplet;
}
...
private Applet applet;
}
```

Ogni volta che raggiunge un punto interessante, l'algoritmo deve fare una pausa in modo che l'utente possa osservare il risultato grafico. Fornite il metodo pause descritto qui e invocatelo in vari punti dell'algoritmo. Il metodo pause ridisegna l'applet e attende dormiente per un breve intervallo di tempo che è proporzionale al numero di passi indicati. Tuttavia, se il thread viene interrotto, il metodo lancia un'eccezione di tipo InterruptedException, che farà terminare il metodo run. (Trasformando la segnalazione di interruzione in un'eccezione, non avete bisogno di aggiungere una verifica di interruzione al codice dell'algoritmo.) Multithreading


```
public void pause(int steps)
throws InterruptedException
{
if (Thread.currentThread().isInterrupted())
throw new InterruptedException();
```

```
applet.repaint();
Thread.sleep(steps * DELAY);
}
```

Dovete, poi, aggiungere alla classe dell'algoritmo un metodo draw che rappresenti graficamente lo stato attuale della struttura dati, evidenziando le entità di particolare interesse. Il metodo draw è specifico di ciascun algoritmo. In questo caso, il metodo draw disegna gli elementi dell'array come una sequenza di barrette di colori diversi: la porzione già ordinata in blu, l'elemento attualmente visitato in rosso, gli elementi rimanenti in nero.

```
public void draw(Graphics2D g2)
{
int deltaX = applet.getWidth() / a.length;
for (int i = 0; i < a.length; i++)
{
if (i == markedPosition)
g2.setColor(Color.red);
else if (i <= alreadySorted)
g2.setColor(Color.blue);
else
g2.setColor(Color.black);
g2.draw(new Line2D.Double(i * deltaX, 0,
i * deltaX, a[i]));
}
}
```

Dovete aggiornare le posizioni speciali mentre l'algoritmo procede e interrompere temporaneamente l'animazione ogni volta che accade qualcosa di interessante. La pausa dovrebbe essere proporzionale al numero di passi che sono stati eseguiti. Per un algoritmo di ordinamento, fate semplicemente un'unità di pausa per ogni elementi dell'array che viene visitato.

Ecco il metodo minimumPosition, prima che venga inserito il codice per l'animazione.

```
public int minimumPosition(int from)
{
int minPos = from;
for (int i = from + 1; i < a.length; i++)
```

```
if (a[i] < a[minPos]) minPos = i;
return minPos;
}
```

Dopo ciascuna iterazione del ciclo for, vogliamo aggiornare le posizioni che indicano lo stato dell'algoritmo; successivamente, vogliamo impostare una pausa per l'applet.

Per misurare in modo equo il costo di ciascun passo, facciamo una pausa di due unità ***30***

Multithreading

di tempo, perché vengono esaminati due elementi dell'array. Il metodo pause potrebbe lanciare una InterruptedException, che quindi dobbiamo dichiarare.

```
public int minimumPosition(int from)
throws InterruptedException
{
int minPos = from;
for (int i = from + 1; i < a.length; i++)
{
if (a[i] < a[minPos]) minPos = i;
markedPosition = i;
pause(2);
}
return minPos;
}
```

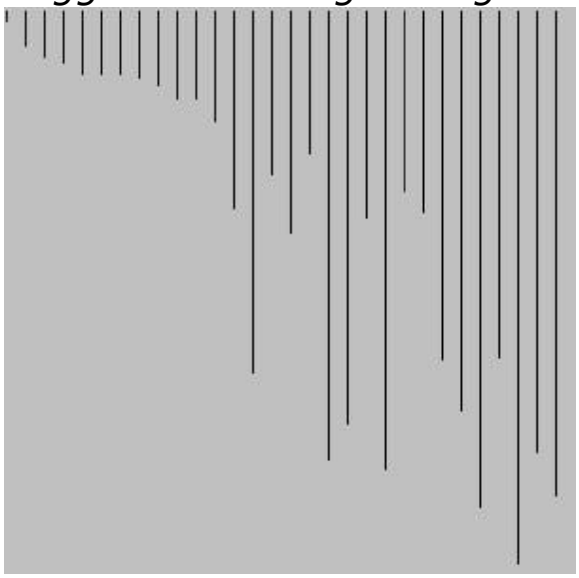
Il metodo sort viene modificato nello stesso modo: troverete il codice alla fine di questo paragrafo. Questo termina le modifiche da apportare alla classe dell'algoritmo, per cui esaminiamo ora la classe dell'applet.

Per prima cosa, abbiamo bisogno di un'interfaccia utente per far partire l'animazione. In questo esempio, la faremo molto semplice: quando l'utente preme il pulsante del mouse all'interno dell'area di visualizzazione dell'applet, l'animazione parte; se un'animazione è già in esecuzione, interrompiamo tale thread e lo facciamo terminare. Questo effetto si ottiene nel solito modo, aggiungendo all'applet un ricevitore di eventi del mouse.

```
public class SelectionSortApplet extends Applet
{
```

```
public SelectionSortApplet()
{
class MousePressListener extends MouseAdapter
{
public void mousePressed(MouseEvent event)
{
if (animation != null && animation.isAlive()) animation.interrupt();
startAnimation();
}
}
MouseListener listener = new MousePressListener(); addMouseListener(listener);
...
animation = null;
}
...
private Thread animation;
}
```

Il metodo paint dell'applet invoca il metodo draw dell'oggetto che esegue l'algoritmo.




```
public void paint(Graphics g)
{
```

```
if (sorter == null) return;
Graphics2D g2 = (Graphics2D)g;
sorter.draw(g2);
}
```

Il metodo startAnimation costruisce un oggetto di tipo SelectionSorter, al quale fornisce un nuovo array e il riferimento all'applet, this. Quindi, il metodo costruisce un thread il cui metodo run chiama il metodo sort dell'oggetto appena costruito.

```
public void startAnimation()
{
class AnimationThread extends Thread
{
public void run()
{
try
{
sorter.sort();
}
catch (InterruptedException exception)
{
}
}
}
int[] values = ArrayUtil.randomIntArray(30, 300); sorter = new SelectionSorter(values, this);
animation = new AnimationThread();
animation.start();
}
```

Figura 3

Un passo nell'animazione dell'algoritmo di ordinamento per selezione

Le classi per l'applet che fa l'animazione e i file HTML per eseguire l'applet si trovano alla fine del paragrafo. Eseguite l'applet e premete il pulsante del mouse all'interno dell'area di visualizzazione dell'applet: l'animazione inizia. Potete eseguire l'animazione finché l'array non è stato ordinato completamente, oppure la potete interrompere in qualsiasi momento. Ogni volta che premete il pulsante del mouse all'interno dell'area di visualizzazione dell'applet, inizia una nuova animazione con un diverso array riempito con numeri casuali (osservate la Figura 3).

L'Esercizio P9 vi chiede di animare l'algoritmo di ordinamento per fusione. Se svol-gete quell'esercizio, fate poi partire entrambi gli applet ed eseguiteli in parallelo, osservando quale algoritmo sia più veloce: in realtà, può darsi che troviate un risultato sorprendente. Se inserite ritardi equi nell'animazione dell'ordinamento per fusione, per tener conto delle azioni di copiatura che coinvolgono l'array temporaneo, troverete che tale algoritmo non ha prestazioni tanto buone per array di piccole dimensioni.

Se, però, aumentate la dimensione dell'array, il vantaggio dell'algoritmo di ordinamento per fusione diventa evidente.

File SelectionSortApplet.html

```
<html>
<body>
<p>Click on the applet to start the animation.</p>
<applet code="SelectionSortApplet.class"
width="300" height="300">
</applet>
</body>
</html>
```

File SelectionSortApplet.java

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.MouseEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseListener;
```

*/**
Questo applet anima l'algoritmo di ordinamento per selezione.
*/
public class SelectionSortApplet extends Applet
{
/**
Costruisce l'applet e imposta il gestore del mouse.
Premendo il pulsante del mouse sull'area di visualizzazione dell'applet si fa partire l'animazione.
*/
public SelectionSortApplet()
{
class MousePressListener extends MouseAdapter
{
public void mousePressed(MouseEvent event)

```
*/
public void startAnimation()
{
class AnimationThread extends Thread
{
public void run()
{
try
{
sorter.sort();
}
catch (InterruptedException exception)
{
}
}
}
int[] values = ArrayUtil.randomIntArray(30, 300); sorter = new SelectionSorter(values, this);
animation = new AnimationThread();
animation.start();
}
private SelectionSorter sorter;
private Thread animation;
}
```

File SelectionSorter.java

```
import java.applet.Applet;
import java.awt.Color;
```



```
import java.awt.Graphics2D;
import java.awt.geom.Line2D;
/**
Questa classe ordina un array, usando l'algoritmo di ordinamento per selezione.
*/
public class SelectionSorter
{
/**
```

```
Costruisce un ordinatore per selezione.
@param anArray l'array da ordinare
@param anApplet l'applet che effettua l'animazione
*/
public SelectionSorter(int[] anArray, Applet anApplet)
{
a = anArray;
applet = anApplet;
}
/**
Ordina l'array gestito da questo ordinatore per selezione.
*/
public void sort()
throws InterruptedException
{
for (int i = 0; i < a.length – 1; i++)
{
int minPos = minimumPosition(i);
swap(minPos, i);
// per l'animazione
alreadySorted = i;
pause(2);
}
}
/**
Cerca l'elemento minore in una coda dell'array.
@param from la prima posizione da considerare nell'array a
@return la posizione dell'elemento minore nella coda a[from]
... a[a.length – 1]
*/
private int minimumPosition(int from)
throws InterruptedException
{
int minPos = from;
for (int i = from + 1; i < a.length; i++)
{
if (a[i] < a[minPos]) minPos = i;
// per l'animazione
```

```
markedPosition = i;
pause(2);
}
return minPos;
}
```

Multithreading

35

```
/**
Scambia due elementi dell'array.
@param i la prima posizione da scambiare
@param j la seconda posizione da scambiare
*/
private void swap(int i, int j)
{
int temp = a[i];
a[i] = a[j];
a[j] = temp;
}
/**
Disegna lo stato attuale dell'algoritmo di ordinamento.
@param g2 il contesto grafico
*/
public void draw(Graphics2D g2)
{
int deltaX = applet.getWidth() / a.length;
for (int i = 0; i < a.length; i++)
{
if (i == markedPosition)
g2.setColor(Color.red);
else if (i <= alreadySorted)
g2.setColor(Color.blue);
else
g2.setColor(Color.black);
g2.draw(new Line2D.Double(i * deltaX, 0,
i * deltaX, a[i]));
}
}
/**
```

Inserisce una pausa nell'animazione.
@param steps il numero di passi di durata della pausa
**/*
public void pause(int steps)
throws InterruptedException
{
if (Thread.currentThread().isInterrupted())
throw new InterruptedException();
applet.repaint();
*Thread.sleep(steps * DELAY);*
}
private int[] a;
// l'applet è necessario per mettere in pausa l'animazione
private Applet applet;
// queste variabili servono per disegnare
private int markedPosition = -1;
private int alreadySorted = -1;
private static final int DELAY = 100;
}

File ArrayUtil.java

import java.util.Random;
*/***
Questa classe contiene metodi utili per
la manipolazione di array.
**/*
public class ArrayUtil
{
*/***
Costruisce un array contenente valori casuali.
@param length la lunghezza dell'array
@param n il numero di valori casuali possibili
@return un array contenente length numeri
casuali compresi fra 0 e n-1
**/*
public static int[] randomIntArray(int length, int n)
{

```
int[] a = new int[length];
Random generator = new Random();
for (int i = 0; i < a.length; i++)
a[i] = generator.nextInt(n);
return a;
}
/**
Stampa tutti gli elementi di un array.
@param a l'array da stampare
*/
public static void print(int[] a)
{
for (int i = 0; i < a.length; i++)
System.out.print(a[i] + " ");
System.out.println();
}
}
```

Note di cronaca 1

Sistemi embedded

Un sistema embedded (incorporato) è un sistema digitale che controlla un dispositivo che consiste in un processore e altri componenti hardware controllati da un programma per calcolatore. Diversamente da ciò che avviene per un personal computer, che è stato progettato per un utilizzo flessibile ed è in grado di eseguire molti programmi diversi, l'hardware e il software di un sistema embedded vengono progettati su misura per un particolare dispositivo. I dispositivi controllati da calcolatori stanno diven-

tando sempre più comuni, spaziando da lavatrici ad apparecchi medicali, motori di automobili e astronavi.

Esistono parecchi problemi specifici che riguardano la programmazione di sistemi embedded, il più importante dei quali è che viene applicato uno standard molto più elevato di controllo di qualità. Spesso i venditori sono poco interessati agli errori presenti nel software dei personal computer, perché vi possono sempre far installare una modifica riparatrice (patch) o aggiornarvi il software a una versione più recente. Tuttavia, nei

sistemi embedded questa opzione non esiste: pochi acquirenti si sentirebbero a proprio agio con l'aggiornamento del software della propria lavatrice e del motore della propria automobile. Se avete mai consegnato un'esercitazione di programmazione che ritenevate corretta, soltanto per fare in modo che il docente vi scoprisse errori, allora sapete quanto sia difficile scrivere software che possa svolgere il proprio compito in modo affidabile per molti anni senza la possibilità di modificarlo.

Gli standard di qualità sono particolarmente importanti nei dispositivi il cui mal-funzionamento può provocare danni a cose o a persone.

Molti acquirenti di personal computer comprano calcolatori veloci, con una grande memoria di massa, perché l'investimento viene ripagato nel tempo eseguendo molti programmi sullo stesso hardware. Al contrario, l'hardware dei sistemi embedded non viene condiviso, è dedicato a un solo dispositivo. Per ogni copia di tale dispositivo viene costruito un diverso processore, con la relativa memoria, e così via. Se si riesco-no a risparmiare anche soltanto pochi centesimi nella costruzione di ciascuna unità, il risparmio aumenta rapidamente per dispositivi che vengono prodotti in grandi volu-mi, per cui i programmatori di sistemi embedded hanno un incentivo economico molto maggiore a utilizzare minori risorse, di quanto non abbiano i programmatori di software per personal computer. Sfortunatamente, cercare di risparmiare risorse rende solitamente più difficile la scrittura di programmi che funzionano correttamente.

Solitamente, i sistemi embedded vengono programmati usando linguaggi di basso livello, per evitare il sovraccarico di un complesso sistema di esecuzione. Il sistema di esecuzione di Java, con i suoi meccanismi di sicurezza, di garbage collection, di supporto per il multithreading, e così via, sarebbe troppo costoso per una lavatrice. Tuttavia, si sono costruiti alcuni dispositivi con una versione ridimensionata di Java: l'ambiente Java 2 Micro Edition. Alcuni esempi sono telefoni cellulari intelligenti e calcolatori di bordo per automobili. La Java 2 Micro Edition è una buona scelta per dispositivi che vengono connessi

in rete e che devono eseguire in modo sicuro nuove applicazioni.

Ad esempio, potete scaricare un programma in un telefono cellulare abilitato all'utilizzo di Java ed essere certi che non potrà provocare danni ad altre parti del software del telefono stesso.

Riepilogo del capitolo

1. Un thread è un'unità del programma che viene eseguita indipendentemente dalle altre porzioni del programma stesso.

2. Quando viene fatto partire un oggetto di tipo Thread, viene eseguito in un nuovo thread il codice del suo metodo run.

3. Il metodo sleep pone in uno stato dormiente il thread attuale per il numero di millisecondi specificato.

4. Quando un thread viene interrotto, la reazione più comune è la terminazione del metodo run.

5. Il pianificatore (scheduler) dei thread esegue ciascun thread per un breve intervallo di tempo, detto time slice (finestra temporale).

6. Un thread termina quando termina il suo metodo run.

7. Un thread dovrebbe, ogni tanto, controllare se è stato interrotto, chiamando il metodo isInterrupted.

8. Si ha una condizione di corsa critica quando l'effetto complessivo di più thread che condividono dati dipende dall'ordine in cui i thread vengono eseguiti.

9. Eseguendo un metodo sincronizzato, un thread diventa titolare del blocco dell'oggetto, dopo di che nessun altro thread può bloccare lo stesso oggetto.

10. Un blocco fatale (deadlock) accade se nessun thread può procedere nell'esecuzione perché ciascun thread è in attesa che un altro thread bloccato produca prima qualche risultato.

11. L'invocazione di wait all'interno di un metodo sincronizzato mette in attesa il thread attuale e consente a un altro thread di bloccare l'oggetto.

12. Un thread che ha invocato wait rimane bloccato finché un altro thread invoca notifyAll o notify sull'oggetto per il quale il thread sta aspettando.

Classi, oggetti e metodi presentati nel capitolo

java.lang.InterruptedException
java.lang.Object
notify
notifyAll
wait
java.lang.Thread
currentThread
isInterrupted
run
sleep
start

Esercizi di ripasso

Esercizio R.1. *Eseguite un programma con le seguenti istruzioni: GreetingThread t1 = new GreetingThread("Hello, World!"); GreetingThread t2 = new GreetingThread("Goodbye, World!"); t1.run();*

t2.run();

Osservate che i thread non vengono eseguiti in parallelo. Fornite spiegazioni.

Esercizio R.2. *Nel programma del Paragrafo 1, è possibile che entrambi i thread si trovino contemporaneamente nello stato dormiente? E che nessuno dei due sia dormiente in un certo istante? Fornite spiegazioni.*

Esercizio R.3. *In Java, un programma con un'interfaccia utente grafica ha più di un thread. Spiegate come è possibile dimostrare ciò.*

Esercizio R.4. *Perché il metodo stop per provocare la terminazione di un thread è sconsigliato? Come potete porre fine a un thread?*

Esercizio R.5. *Fornite un esempio del motivo per cui si può volere la terminazione di un thread.*

Esercizio R.6. *Supponete di circondare ciascuna invocazione del metodo sleep con un blocco try/catch che intercetti l'eccezione di tipo InterruptedException e la ignori.*

Che problemi si creano?

Esercizio R.7. *Supponete che esistano i seguenti thread.*

Thread
Stato
Priorità
Thread-0
Eseguibile
5
Thread-1
Dormiente
5
Thread-2
Eseguibile
5
Thread-3
In attesa
5

Il pianificatore sta per assegnare una nuova finestra temporale a un thread. Quale sceglie?

Esercizio R.8. *Supponete che esistano i seguenti thread.*

Thread
Stato
Priorità
Thread-0
Eseguibile
5
Thread-1
Dormiente
6
Thread-2
Eseguibile
5
Thread-3
In attesa
6
Thread-4
Eseguibile
4

Il pianificatore sta per assegnare una nuova finestra temporale a un thread. Quale sceglie?

40

Multithreading

Esercizio R.9. *Cos'è una condizione di corsa critica? Come potete evitarla?*

Esercizio R.10. *Cos'è una condizione di blocco fatale (deadlock)? Come potete evitarla?*

Esercizio R.11. *Qual è la differenza tra un thread che si trova nello stato dormiente per aver invocato il metodo sleep e un thread che si trova in attesa per aver invocato il metodo wait?*

Esercizio R.12. *Cosa succede se un thread invoca wait e nessun altro thread invoca notifyAll o notify?*

Esercizio R.13. *Come potete modificare i thread di Argomenti avanzati 4 in modo da evitare il blocco fatale?*

Esercizio R.14. *Nel programma di animazione dell'algoritmo del Paragrafo 4 non abbiamo usato wait e notify. Perché?*

Esercizio R.15. *Considerate il programma di animazione dell'algoritmo del Paragrafo 4. Supponete di premere il pulsante del mouse dopo che l'animazione è iniziata ma prima che sia terminata: in tal caso, il thread di animazione viene interrotto. Supponete che il thread sia in quel momento dormiente. Spiegate come avviene la terminazione del suo metodo run. Al contrario, se quando viene interrotto non è dormiente, come avviene la terminazione del suo metodo run?*

Esercizi di programmazione

Esercizio P.1. *Realizzate una classe Queue (coda) i cui metodi add e remove siano sincronizzati. Fornite un thread, chiamato produttore, che continua a inserire stringhe nella coda finché ci sono meno di 10 elementi in essa. Quando la coda diventa troppo piena, il thread aspetta. Come stringhe, usate semplicemente le indicazioni orarie fornite da new Date().toString(). Fornite poi un secondo thread, detto consumatore, che continua a rimuovere stringhe dalla cosa, stampandole, finché la coda non è vuota. Quando la coda è*

vuota, il thread aspetta. Sia il thread produttore che il thread consumatore devono eseguire 1000 iterazioni.

Esercizio P.2. *Migliorate l'esercizio precedente usando un numero variabile di thread produttori e consumatori. Chiedetene il numero all'utente del programma.*

Esercizio P.3. *Realizzate la classe Purse (borsellino) del Capitolo 12 in modo che il metodo addCoin sia sincronizzato, e realizzate anche un metodo removeCoin. Implementate un thread che aggiunge monete da un penny, un altro thread che aggiunge monete da un quarto di dollaro, e un altro thread che rimuove e visualizza le monete.*

Esercizio P.4. *Modificate l'applet che visualizza automobili visto nel Capitolo 4 in modo che le automobili si muovano. Usate un diverso thread per ciascuna automobile.*

Esercizio P.5. *Modificate l'esercizio precedente in modo che le automobili cambino direzione quando colpiscono un bordo della finestra.*

Esercizio P.6. *Scrivete un programma WordCount che conti le parole in uno o più file.*

Fate partire un nuovo thread per ciascun file. Ad esempio, se lo chiamate con java WordCount report.txt address.txt Homework.java il programma potrebbe stampare

address.txt: 1052

Homework.java: 445

report.txt: 2099

Esercizio P.7. *Scrivete un programma Find che cerchi in tutti i file specificati sulla riga dei comandi e che stampi tutte le righe che contengono una determinata parola. Fate partire un nuovo thread per ciascun file. Ad esempio, se lo chiamate con java Find Buff report.txt address.txt Homework.java il programma potrebbe stampare*

report.txt: Buffet stype lunch will be available at the address.txt: Buffet, Warren|11801 Trenton Court|Dallas|TX

Homework.java: BufferedReader in;

address.txt: Walters, Winnie|59 Timothy Circle|Buffalo|MI ***Esercizio P.8.*** *Invece di usare un thread e un metodo pause,*

usate la classe Timer del pacchetto Swing per animare un algoritmo. Ogni volta che il temporizzatore emette un evento di azione, eseguite l'algoritmo fino al passo successivo e visualizzate lo stato. Questo richiede una modifica più estesa all'algoritmo. Dovete implementare un metodo runToNextStep che possa eseguire l'algoritmo un passo alla volta. Aggiungete all'algoritmo sufficienti variabili istanza per tenere traccia del punto in cui era arri-vato il passo precedente. Ad esempio, nel caso dell'algoritmo di ordinamento per selezione, se conoscete i valori di alreadySorted e di markedPosition, potete eseguire il passo successivo.

Esercizio P.9. *Realizzate un'animazione per l'algoritmo di ordinamento per fusione visto nel Capitolo 16. Implementate nuovamente l'algoritmo in modo che invocazioni ricorsive ordinino gli elementi all'interno di una porzione dell'array originario, invece che in propri array temporanei:*

```
public void mergeSort(int from, int to)
{
if (from == to) return;
int mid = (from + to) / 2;
mergeSort(from, mid);
mergeSort(mid + 1, to);
merge(from, mid, to);
}
```


Il metodo merge fonde le porzioni ordinate a[from]...a[mid] e a[mid + 1]...a[to].

Fate una pausa nel metodo merge ogni volta che esaminate un elemento dell'array.

Colorate in blu la porzione a[from]...a[to] e in rosso l'elemento in esame.

Esercizio P.10. *Realizzate l'algoritmo di ordinamento per fusione visto nel Capitolo 16 facendo partire un nuovo thread per ciascun oggetto di tipo MergeSorter più piccolo.*

Programmazione di rete

Obiettivi del capitolo

- *Capire il concetto di socket*

■ Imparare a inviare e ricevere dati tramite i socket

■ Implementare programmi client e server di rete

■ Comunicare con server Web e applicazioni sul lato server mediante il protocollo HTTP (Hypertext Transfer Protocol)

2

Programmazione di rete

I*nternet è un insieme di reti mon-*

Probabilmente avete già un po' di esperienza di Internet: la rete globale diali, di apparecchiature di instra-che collega tra loro milioni di computer. In particolare, usate Internet damento e di computer che usa-ogni volta che navigate nel World Wide Web, ma fate attenzione perché no un insieme comune di protocolli per definire come ciascuna

dire Internet non è come dire "il Web". Il World Wide Web è soltanto parte della rete interagisce con

uno dei molti servizi che vengono offerti sulla rete Internet. La posta le altre componenti.

elettronica (email) è un altro servizio molto diffuso e anch'esso usa Internet, ma la sua implementazione è diversa da quella del Web. In questo capitolo vedrete cosa succede in realtà quando inviate un messaggio di posta elettronica o quando recuperate una pagina Web da un server remoto. Imparerete anche a scrivere programmi che sono in grado di ricevere dati da siti sparsi in Internet e a scrivere programmi server che possono fornire informazioni ad altri programmi.

1

Il protocollo Internet

I computer possono essere connessi l'uno all'altro mediante una grande varietà di mezzi fisici. In un laboratorio di informatica, ad esempio, i computer sono connessi con un cablaggio di rete: gli impulsi elettrici che rappresentano l'informazione fluiscono lungo i cavi. Se usate un modem per connettere il vostro computer a Internet, i segnali viaggiano attraverso una comune linea telefonica, codificati come toni. In una rete wireless (senza fili), i segnali vengono inviati trasmettendo un segnalo modulato a radiofrequenza. Le caratteristiche fisiche di queste trasmissioni sono fortemente

diverse, ma alla fin fine consistono nell'invio e nella ricezione di flussi di zeri e di uni lungo la connessione di rete.

Questi zeri e uni rappresentano due tipi di informazioni: dati di applicazioni, i dati che un computer vuole veramente inviare a un altro, e dati dei protocolli di rete, i dati che descrivono come si possa raggiungere il destinatario desiderato e come verificare la presenza di errori e di perdite di dati nella trasmissione. I dati dei protocolli seguono regole predeterminate in un particolare protocollo di rete. Diversi protocolli di rete vengono comunemente utilizzati, come Microsoft Networking, Novell Netware o Ap-pleTalk: questi protocolli sono adatti a reti locali (LAN, Local Area Network, si veda Note di cronaca 1). Il protocollo Internet (IP, Internet Protocol), invece, è stato sviluppato per permettere a diverse reti locali di comunicare fra loro ed è diventato la base per la connessione tramite Internet di computer sparsi nel mondo. In questo capitolo discuteremo il protocollo IP.

Supponete che il computer A voglia inviare dati al computer B, entrambi su Internet. I computer non sono direttamente connessi con un cavo, come potrebbe accadere se si trovassero entrambi nella rete locale; al contrario, A può essere il computer a casa di qualcuno ed essere connesso tramite il telefono a un fornitore di servizi Internet (ISP, Internet Service Provider), il quale a sua volta è connesso a un punto di accesso a Internet (Internet access point); B potrebbe essere un computer di una rete locale appartenente a una grande organizzazione avente un proprio punto di accesso a Internet, che si può trovare dall'altra parte del mondo rispetto ad A. Internet stessa, infine, è un complicato insieme di percorsi sui quali un messaggio può viaggiare da un punto di accesso a un altro (osservare la Figura 1). Tali connessioni possono trasportare milioni di messaggi, non solo i dati che A invia a B.

Figura 1
Due computer che comunicano attraverso Internet

Perché i dati possano arrivare a destinazione, devono essere contrassegnati con l' indirizzo di destinazione. Nel protocollo IP, gli indirizzi vengono indicati con sequenze di quattro numeri, ciascuno che rappresenta un byte (cioè, un numero compreso tra 0

e 255); ad esempio, 130.65.86.66. (Poiché non ci sono sufficienti indirizzi di 4 byte per tutti i dispositivi che si vorrebbe connettere a Internet, questi indirizzi verranno estesi a 6 byte in un prossimo futuro.) Per inviare dati, A deve conoscere l'indirizzo Internet di B e includerlo nella porzione riguardante il protocollo quando invia i dati attraverso Internet. Il software di instradamento (routing) distribuito lungo la rete può, quindi, consegnare i dati a B.

Ovviamente, indirizzi come 130.65.86.66 non sono facili da ricordare. Non sareste contenti se doveste usare sequenze di numeri ogni volta che inviate un messaggio di posta elettronica o richiedete informazioni a un server Web. Su Internet, i computer possono avere ciò che viene chiamato nome di dominio, che è più facile da ricordare, come cs.sjsu.edu o java.sun.com. Uno speciale servizio, detto Domain Naming Service (DNS), traduce i nomi di dominio in indirizzi Internet. In questo modo, se il computer A vuole avere informazioni da java.sun.com, per prima cosa chiede al DNS di tradurre questo nome di dominio in un indirizzo Internet numerico, quindi inserisce tale indirizzo numerico nella richiesta.

Un aspetto interessante del protocollo IP è che grandi quantità di dati possono essere frammentati in pacchetti più maneggevoli. Ciascun pacchetto viene consegnato indipendentemente dagli altri, e diversi pacchetti che fanno parte della stessa trasmissione possono prendere strade diverse lungo la rete. I pacchetti sono numerati e il destinatario li rimette insieme nell'ordine corretto.

Il protocollo Internet ha una sola funzione: tentare di consegnare i dati da un computer a un altro attraverso Internet. Se alcuni dati vengono perduti o modificati durante il processo di trasferimento, il protocollo IP ha dei meccanismi di salvaguardia al suo interno per essere certi che il destinatario sia consapevole di tale caso sfortunato ***4***

e non si affidi ciecamente a dati incompleti. Tuttavia, IP non ha modo di segnalare al mittente la necessità di ritentare la trasmissione: questo è compito di un protocollo di più alto livello, il Transmission Control Protocol (TCP). Questo protocollo ha l'obiettivo di una consegna affidabile dei dati, con nuovi tentativi se si verificano errori, e segnala al mittente se l'obiettivo è stato raggiunto oppure no. La maggior parte dei programmi di Internet, ma non tutti, usano il protocollo TCP per la consegna affidabile. (Tra le eccezioni, ci sono i servizi di "flusso multimediale", che scavalcano il più lento protocollo TCP per usare una trasmissione più veloce, tollerando occasionali perdite di informazioni. Tuttavia i servizi Internet più diffusi, come il World Wide Web e la posta elettronica, usano TCP.) Il protocollo TCP è indipendente dal protocollo IP: in linea di principio potrebbe essere utilizzato con un diverso protocollo di rete di più basso livello, ma, in pratica, TCP basato su IP è la combinazione usata più comunemente, e viene spesso chiamata TCP/IP. In questo capitolo concentreremo la nostra attenzione sulle reti TCP/IP.

TCP/IP è l'abbreviazione di Tran-

Un computer connesso a Internet può avere programmi per molti smission Control Protocol su

scopi diversi. Ad esempio, un computer può eseguire sia un server Web Internet Protocol, la coppia di pro-sia un server di posta elettronica. Quando i dati vengono inviati a quel tocolli di comunicazione usata per

computer, devono essere contrassegnati in modo da poter essere con-stabilire una trasmissione affida-

bile di dati fra due computer di

segnati al programma giusto. Il protocollo IP usa i numeri di porta per Internet.

questo scopo: un numero di porta è un numero intero compreso tra 0 e 65535. Il computer che invia i dati deve conoscere il numero di porta del programma destinatario e inserirlo nei dati trasmessi. Alcune applicazioni usano numeri di porta "ben conosciuti": ad esempio, per convenzione, i server Web usano la porta 80, mentre i server di posta elettronica che

usano il protocollo POP (Post Office Protocol) usano la porta 110. I pacchetti TCP, quindi, devono contenere:

- *l'indirizzo Internet del destinatario*
- *il numero di porta del destinatario*
- *l'indirizzo del mittente*
- *il numero di porta del mittente*

Una connessione TCP richiede gli

Potete pensare a una connessione TCP come a una "conduttura" (pipe) indirizzi Internet e i numeri di

fra due computer che connette insieme due porte: i dati fluiscono in porta di entrambe le sue termi-entrambe le direzioni lungo la conduttura. In situazioni reali di pro-nazioni.

grammazione, stabilite semplicemente una connessione e inviate dati lungo di essa, senza preoccuparvi dei dettagli e dei meccanismi di funzionamento di TCP/IP. Vedrete come stabilire una tale connessione nel Paragrafo 3.

Note di cronaca 1

Il modello OSI

Nel 1984, la International Organization for Standardization (ISO) pubblicò una descrizione teorica di protocolli di rete denominata: modello di riferimento per l'inter-connessione di sistemi aperti (Open Systems Interconnection, OSI). Il modello OSI descrive i vari strati di astrazione per il software e l'hardware di rete, e detta regole che Programmazione di rete

Figura 2

Il modello

di riferimento OSI

riguardano il modo in cui ciascuno strato comunica con gli altri. Come mostra la Figura 2, gli strati formano una pila (stack) e ogni strato usa soltanto i servizi dello strato immediatamente sottostante. Questo schema di progettazione protegge gli strati so-prastanti dai dettagli realizzativi degli strati sottostanti.

Ecco una descrizione dettagliata degli strati.

- *Il livello fisico (Physical) ha a che fare con la trasmissione di bit attraverso un supporto fisico come un cavo di rame, una fibra ottica o un collegamento wireless.*

■ *Il livello di connessione dei dati (Data Link) raggruppa i bit dei dati in pacchetti più grandi che vengono trasmessi unitariamente, e assicura che tali pacchetti vengano consegnati senza errori di trasmissione.*

■ *Il livello di rete (Network) gestisce le connessioni attraverso la rete e si occupa, ad esempio, di instradare i pacchetti e di evitare la congestione.*

■ *Il livello di trasporto (Transport) assicura che i pacchetti vengano consegnati da un computer a un altro, richiede la ritrasmissione dei pacchetti perduti e risistema i pacchetti che arrivano in ordine errato.*

■ *Il livello di sessione (Session) stabilisce sessioni di connessione tra due computer e decide quale entità trasmette i dati e per quanto tempo.*

■ *Il livello di presentazione (Presentation) implementa servizi standard di presentazione dei dati, come, ad esempio, le conversioni di formato, la compressione e la cifratura dei dati.*

■ *Il livello di applicazione (Application) si occupa delle applicazioni di rete, quali i browser Web e la posta elettronica.*

Il modello OSI è un modello teorico a cui le reali implementazioni delle reti si ade-guano in vario modo. Nel caso di Internet, il protocollo IP si trova al livello 3 e il protocollo TCP al livello 4; infatti, il protocollo IP è responsabile della consegna dei pacchetti attraverso la rete. Il compito di IP è quello di garantire che i pacchetti siano ***6***

consegnati integri e al corretto destinatario, oppure che non siano consegnati affatto.

Il protocollo TCP ha la responsabilità di garantire la consegna di un flusso di pacchetti: se lo strato TCP si accorge che manca un pacchetto, usa lo strato IP (ma non uno strato inferiore) per chiedere al mittente la ritrasmissione del pacchetto mancante.

2

I protocolli al livello di applicazione

HTTP (Hypertext Transfer Proto-

Nel paragrafo precedente avete visto come l'architettura TCP/IP possa col) è il protocollo che definisce

stabilire una connessione Internet fra due porte di due computer in modo la comunicazione fra browser

che i due computer si possano scambiare dati. Ogni applicazione Inter-Web e server Web.

net usa un diverso protocollo di applicazione, che descrive come vengono trasmessi i dati per quella particolare applicazione.

Un URL (Uniform Resource Lo-

Considerate, ad esempio, il protocollo HTTP (Hypertext Transfer Proto-cator) è un riferimento a una ri-

col), usato per il World Wide Web. Ipotizzate di digitare un indirizzo sorsa informativa (come una pa-Web (detto URL, Uniform Resource Locator, e spesso pronunciato come gina Web o un'immagine) che si

"earl"), come http://java.sun.com/index.html, all'interno della fine-trova nel World Wide Web.

stra del vostro browser e di chiedere al browser di caricare la pagina. Il browser esegue, quindi, i seguenti passi:

1. Esamina la parte di URL compresa fra la doppia barra e la prima barra singola ("java.sun.com"), che identifica il computer a cui vi volete connettere. Poiché questa parte di questo URL contiene delle lettere, deve essere un nome di dominio e non un indirizzo Internet, per cui il browser invia a un server DNS una richiesta per ottenere l'indirizzo Internet del computer il cui nome di dominio è java.sun.com.

2. Dal prefisso http: dell'URL, il browser deduce che volete usare il protocollo HTTP, la cui porta predefinita è la porta 80.

3. Stabilisce una connessione TCP/IP con la porta 80 all'indirizzo Internet ottenuto al passo 1.

4. Dal suffisso /index.html, deduce che volete vedere il file /index.html, per cui invia una richiesta, composta come comando HTTP, attraverso la connessione stabi-lita al passo 3. La richiesta è così fatta:

GET /index.html HTTP/1.0

riga vuota

5. Il server Web in esecuzione sul computer il cui indirizzo Internet corrisponde a quello ottenuto dal browser al passo 1

riceve la richiesta e la decodifica. Quindi, recupera il file /index.html e lo invia al browser del vostro computer.

6. Il browser visualizza il contenuto del file, in modo che lo possiate vedere. Poiché si tratta di un file HTML, il browser traduce i codici HTML in font, punti elenco, linee di separazione, e così via. Se il file HTML contiene immagini, il browser invia ulteriori richieste di tipo GET, una per ciascuna immagine, attraverso la stessa connessione, per recuperare i dati delle immagini.

Potete eseguire l'esperimento seguente per vedere in azione questo procedimento. Il programma telnet consente a un utente di digitare caratteri che vengono inviati a un computer remoto e visualizza i caratteri che il computer remoto invia in risposta. Nel

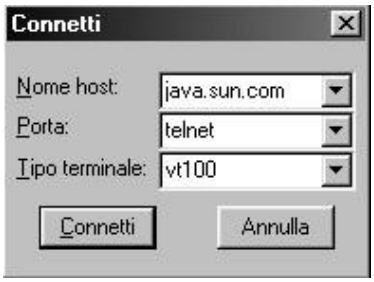


Figura 3
Usare telnet
per connettersi
a un server Web

sistema Windows, il programma Telnet è solitamente già installato, ma probabilmente non lo vedrete sul vostro schermo; provate a premere il pulsante Start (Avvio), selezionare "Run..." ("Esegui...") e digitate telnet nella finestra "Open" ("Esegui"). In un sistema Macintosh, probabilmente dovrete scaricare e installare il programma NCSA Telnet all'indirizzo http://www.ncsa.uiuc.edu/SDG/Software/MacTelnet/. I sistemi UNIX (e Linux) hanno normalmente telnet già installato. Per questo esperimento, dovete far partire telnet usando java.sun.com come host e 80 come porta. I dettagli dipendono dal vostro programma telnet. Se eseguite il programma dalla riga dei comandi, digitate semplicemente

telnet java.sun.com 80

Con alcune versioni di telnet, dovete inserire i valori di host e di porta in una finestra di dialogo. La Figura 3 mostra tale finestra nella versione Windows.

Dopo aver fatto partire il programma, digitate con molta attenzione, senza fare alcun errore e senza premere il tasto di cancellazione, quanto segue: GET /index.html HTTP/1.0

Quindi, premete due volte il tasto Invio (Enter).

Il programma telnet è uno stru-

Non vedrete ciò che digitate, perché il programma telnet invia le vo-mento utile per stabilire connes-

stre digitazioni direttamente al server. (Alcune versioni di telnet hanno sioni di prova con i server.

un'opzione di "eco locale" che mostra ciò che digitate. Se il vostro ce l'ha, abilitarla per questo esperimento è una buona idea.) Il server ora invia una risposta alla richiesta: osservate la Figura 4. La risposta, ovviamente, consiste nel file index.html, ma il programma telnet non è un browser e non capisce i marcatori HTML, per cui visualizza semplicemente il file HTML: testo e marcatori insieme.

Il comando GET è uno dei comandi del protocollo HTTP. La tabella 1 mostra gli altri comandi del protocollo, che, come potete vedere, è piuttosto semplice.

Il comando HTTP GET richiede in-

A ogni modo, cercate di non confondere HTML con HTTP. HTML è formazioni a un server Web, il

un formato di documento (con comandi come <h1> o <ul>) che descrive la quale restituisce la risorsa richie-struttura di un documento, con titoli, elenchi puntati, immagini, colle-sta, che può essere una pagina

gamenti ipertestuali, e così via. HTTP è un protocollo (con comandi come Web, un'immagine, o altri dati.

GET e POST) che descrive l'insieme di comandi per le richieste a un server Web. I browser Web sanno come visualizzare documenti HTML e come inviare comandi HTTP. I server Web non sanno nulla di HTML: capiscono sol-

```
Telnet - java.sun.com
Connetti  Modifica  Terminale  ?
ght="1" bgcolor="#cccccc"><img src="/images/v3_pixel.gif" width="1" height="1"><
/td>
    </tr>

         <tr>
             <td width="1" height="24" bgcolor="#cccccc"><img src="/images/v3_pi
xel.gif" width="1" height="24"></td>
                                     <td colspan="5" height="24"></td>
                                                                    <td width="
1" height="24" bgcolor="#cccccc"><img src="/images/v3_pixel.gif" width="1" heigh
t="24"></td>
            </tr>

                 <tr>
                     <td colspan="7" height="1" bgcolor="#cccccc"><img src="/ima
ges/v3_pixel.gif" width="1" height="1"></td>
                                             </tr>
                                                  </table>
                                                          </span>


                                                                    </BODY>

</HTML>
```

8

Programmazione di rete

Figura 4

Il server Web invia la risposta

Tabella 1

Comandi HTTP

Comando

Significato

GET

Richiede una risorsa

HEAD

Richiede soltanto le informazioni contenute nell'intestazione di una risorsa OPTIONS

Richiede le opzioni di comunicazione di una risorsa POST

Fornisce dati in ingresso per un comando eseguito sul server e ne ottiene il risultato

PUT

Memorizza una risorsa sul server

DELETE

Cancella una risorsa dal server

TRACE

Conserva un tracciato della comunicazione con il server tanto il protocollo HTTP e sanno come recuperare le risorse richieste, che possono essere documenti HTML, immagini GIF o JPEG, o qualsiasi altro dato che possa essere visualizzato in un browser Web.

HTTP è soltanto uno dei molti protocolli di applicazione usati in Internet. Un altro protocollo usato comunemente è il Post Office Protocol (POP), per scaricare da un server di posta elettronica i messaggi ricevuti. Per inviare messaggi, usate invece un altro protocollo, chiamato Simple Mail Transfer Protocol (SMTP). Non vogliamo entrare nei dettagli di questi protocolli, ma la Figura 5 vi da un'idea dei comandi usati dal protocollo POP.

```
+OK San Quentin State POP server
USER harryh
+OK Password required for harryh
PASS secret
+OK harryh has 2 messages (320 octets)
STAT
+OK 2 320
RETR 1
+OK 120 octets
qui viene incluso il messaggio
DELE 1
+OK message 1 deleted
QUIT
+OK POP server signing off
```


Figura 5
Un esempio di sessione POP (in grassetto le risposte del server)

Un programma client

In questo paragrafo vedrete come scrivere un programma Java che stabilisce una connessione TCP/IP con un server, inviando poi una richiesta e visualizzando le risposta.

Usando la terminologia TCP/IP, a ogni estremo della comunicazione esiste un socket (osservate la Figura 6). In Java, un client crea un socket con la seguente chiamata Socket s = new Socket(nomeHost, numeroPorta);

Ad esempio, per connettersi alla porta HTTP del server java.sun.com, dovete scrivere final int HTTP_PORT = 80;

Socket s = new Socket("java.sun.com", HTTP_PORT); Un socket è un oggetto che in-Il costruttore di socket lancia un'eccezione di tipo UnknownHostExcep-capsula una connessione TCP/IP.

tion se non riesce a trovare il computer a cui connettersi.

Per comunicare con l'altra estre-

Dopo aver creato il socket, ne ottenete i flussi entranti e uscenti: mità della connessione, usate i

flussi di ingresso e uscita con-

InputStream in = s.getInputStream();

nessi al socket.

OutputStream out = s.getOutputStream();

Figura 6

Socket client

e server

10

Programmazione di rete

Quando la trasmissione attraverso

Quando inviate dati al flusso di uscita out, il socket li manda automati-un socket è terminata, ricorda-

camente al server. Il socket cattura la risposta del server, dopodiché la tevi di chiudere il socket stesso.

potete leggere attraverso il flusso in ingresso in (osservate la Figura 6).

Quando avete terminato la comunicazione con il server, dovreste chiudere il socket:

s.close();

In protocolli di tipo testo, trasfor-

Nel Capitolo 14, avete visto che le classi InputStream e OutputStream mate i flussi dei socket in lettori

vengono usate per leggere e scrivere byte. Se volete comunicare con il e scrittori.

server inviando e ricevendo testo, dovreste trasformare tali flussi in lettori e scrittori, nel modo seguente:

BufferedReader reader = new BufferedReader(

new InputStreamReader(in));

PrintWriter writer = new PrintWriter(out);

Uno scrittore di stampa (cioè di tipo PrintWriter) bufferizza i caratteri che gli inviate, cioè i caratteri non vengono inviati immediatamente alla loro destinazione: vengono, invece, inseriti in un array. Quando l'array è pieno, lo scrittore di stampa invia a destinazione tutti i caratteri dell'array. Il vantaggio della bufferizzazione è un miglioramento di prestazioni: dato che serve un po' di tempo per contattare il destinatario e inviare i dati, è costoso impiegare tale tempo di contatto per ogni carattere. Tuttavia, quando si comunica con un server che risponde a richieste, volete essere certi che il server riceva a un certo punto una richiesta completa. Quindi, avete bisogno di svuotare (flush) il buffer manualmente, ogni volta che inviate un comando: writer.print(comando);

writer.flush();

Alla fine di ogni comando svuo-

Il metodo flush svuota il buffer e invia a destinazione tutti i caratteri in tate il buffer dello scrittore con-attesa.

nesso a un socket. In questo

Il programma WebGet al termine di questo paragrafo vi consente di modo il comando viene effetti-recuperare qualsiasi risorsa da un server Web, specificando sulla riga di vamente inviato al server, anche

se il buffer non è pieno.

comando il nome del server e la risorsa. Ad esempio: java WebGet java.sun.com index.html

Il programma stabilisce semplicemente una connessione con il computer desiderato, invia un comando GET e riceve i dati dal server finché il server stesso non chiude la connessione.

File WebGet.java

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;
```



```
/**
Questo programma mostra come usare un socket per comunicare con un server Web. Fornite sulla riga di comando il nome del server e la risorsa, come nell'esempio seguente:
java WebGet java.sun.com index.html
*/
public class WebGet
{
public static void main(String[] args)
throws IOException
{
// leggi gli argomenti della riga comandi
if (args.length != 2)
{
System.out.println(
"usage: java WebGet host resource");
System.exit(0);
}
String host = args[0];
String resource = args[1];
// apri il socket
final int HTTP_PORT = 80;
Socket s = new Socket(host, HTTP_PORT);
// ottieni i flussi
InputStream in = s.getInputStream();
OutputStream out = s.getOutputStream();
// trasforma i flussi in lettori e scrittori
BufferedReader reader = new BufferedReader(
new InputStreamReader(in));
```

```
PrintWriter writer = new PrintWriter(out);
// invia il comando
String command = “GET /” + resource
+ “ HTTP/1.0\n\n”;
writer.print(command);
writer.flush();
// leggi la risposta del server
boolean done = false;
while (!done)
{
String input = reader.readLine();
if (input == null) done = true;
else System.out.println(input);
}
```



```
// chiudi sempre il socket al termine
s.close();
}
}
```


Un programma server

Ora che avete visto come scrivere un client di rete, passeremo al lato server. In questo paragrafo svilupperemo un programma server che permette ai client di gestire un insieme di conti bancari in una banca.

Ogni volta che sviluppate un’applicazione server, dovete specificare un protocollo al livello applicazione che i client devono usare per interagire con il server. Per questo esempio, creeremo un “Simple Bank Access Protocol”, un protocollo semplificato per l’accesso a una banca. La Tabella 2 mostra il formato del protocollo, che è, ovviamente, soltanto un giocattolo per mostrarvi come realizzare un server.

Tabella 2

Un semplice protocollo per l’accesso a una banca **Richiesta del client**

Risposta del server

Significato

BALANCE n
n e il saldo
Leggi il saldo del conto n
DEPOSIT n a
n e il nuovo saldo
Versa la somma a nel conto n
WITHDRAW n a
n e il nuovo saldo
Preleva la somma a dal conto n
QUIT
Nessuna
Termina la connessione

Il programma server attende che i client si connettano a un particolare porta: per questo servizio abbiamo scelto la porta 8888. Per porsi in ascolto di connessioni entranti, si usa un socket di tipo server. Per costruire un socket di tipo server, dovete indicare un numero di porta.

ServerSocket server = new ServerSocket(8888);

Il metodo accept della classe ServerSocket aspetta connessioni di client. Quando un client si connette, il programma server apre un socket con cui comunica con il client.

Socket s = server.accept();

La classe ServerSocket viene

Ora, ottenete i flussi di ingresso e uscita di s esattamente come visto nel usata dalle applicazioni server per

paragrafo precedente:

restare in attesa di connessioni

da parte dei client.

BufferedReader in = new BufferedReader(
new InputStreamReader(s.getInputStream()));
PrintWriter out = new PrintWriter(
s.getOutputStream());

Poi, il server legge un comando dal client:

String line = in.readLine();

Se la variabile line vale null (cioè il client si è disconnesso) o contiene il comando QUIT, allora il server chiude il socket s. Il socket di tipo server non viene mai chiuso: il server continua ad attendere connessioni.

Per analizzare ulteriormente il comando del client, usiamo uno scompositore di stringhe. Il primo token è il comando, e il secondo token è il numero del conto: StringTokenizer tokenizer = new StringTokenizer(line); String command = tokenizer.nextToken();

int account = Integer.parseInt(tokenizer.nextToken()); Se il comando è DEPOSIT, effettuiamo il versamento.

double amount = Double.parseDouble(tokenizer.nextToken()); bank.deposit(account, amount);

Il comando WITHDRAW viene gestito allo stesso modo.

Infine, inviamo al client il numero di conto e il nuovo saldo.

out.println(account + " " + bank.getBalance(account)); Notate che il programma server non termina mai: quando non vi serve più, dovete interromperlo. Ad esempio, se l'avete posto in esecuzione in una finestra di shell, premete Ctrl+C.

Per provare il programma, eseguite il server, quindi usate il programma telnet per connettervi a localhost, numero di porta 8888. Iniziate a scrivere comandi; ecco un tipico dialogo:

BALANCE 5
5 0
DEPOSIT 5 2000
5 2000
WITHDRAW 5 1000
5 1000
QUIT

Osservate la Figura 7.

In alternativa, potete usare un programma client che si connetta al server; ne troverete un esempio alla fine di questo paragrafo.

Questo server ha un'importante limitazione: in ogni istante vi può essere un solo client connesso. Questo limite, nella realtà, sarebbe molto grave: solitamente in un certo istante ci sono molti client che si connettono a un server. Per realizzare tale

comportamento, i programmi server fanno partire un nuovo thread ogni volta che un client si connette. Più thread possono essere eseguiti in parallelo e ciascun thread ha il compito di servire un client. L'Esercizio P.5 propone l'implementazione di un server che usa il multithreading.

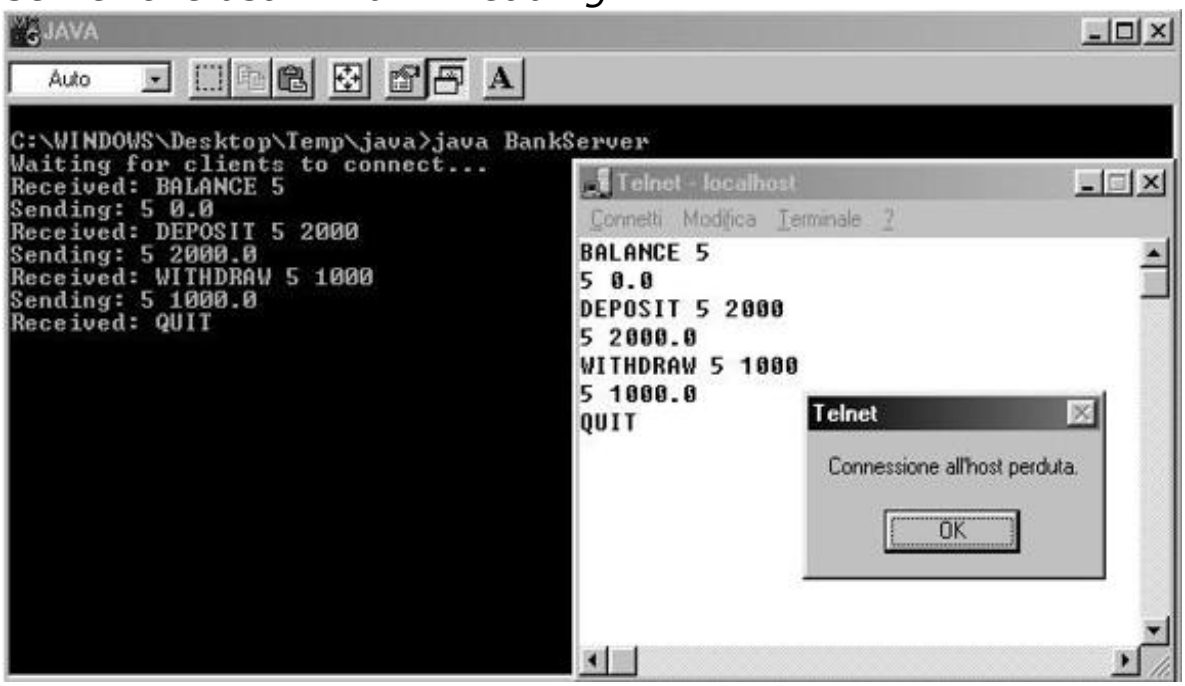


Figura 7
Uso del programma telnet per connettersi al server di una banca

File BankServer.java

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
/**
Un server che esegue il Simple Bank Access Protocol.
*/
public class BankServer
{
public static void main(String[] args)
throws IOException
{
```

final int ACCOUNTS_LENGTH = 10;
Bank bank = new Bank(ACCOUNTS_LENGTH);
final int SBAP_PORT = 8888;
ServerSocket server = new ServerSocket(SBAP_PORT); System.out.println("Waiting for clients to connect…"); while (true)
{
Socket s = server.accept();
BankService service = new BankService(s, bank);
service.doService();
s.close();
}
}
}
Programmazione di rete
15
File BankService.java
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.StringTokenizer;
*/**
Esegue comandi del Simple Bank Access Protocol
attraverso un socket.
**/*
public class BankService
{
*/**
Costruisce un oggetto di servizio che analizza i comandi inviati a una banca attraverso un socket.
@param aSocket il socket
@param aBank la banca
**/*
public BankService(Socket aSocket, Bank aBank)

```
{
s = aSocket;
bank = aBank;
}
/**
Esegue tutti i comandi fino al comando QUIT o alla fine del flusso di ingresso.
*/
public void doService() throws IOException
{
BufferedReader in = new BufferedReader(
new InputStreamReader(s.getInputStream()));
PrintWriter out = new PrintWriter(
s.getOutputStream());
while (true)
{
String line = in.readLine();
System.out.println("Received: " + line);
if (line == null || line.equals("QUIT"))
return;
String response = executeCommand(line);
System.out.println("Sending: " + response);
out.println(response);
out.flush();
}
}
```



```
/**
Esegue un singolo comando.
@param line il comando
@return la risposta da inviare al client
*/
public String executeCommand(String line)
{
StringTokenizer tokenizer = new StringTokenizer(line); String command = tokenizer.nextToken();
```

```
int account = Integer.parseInt(tokenizer.nextToken()); if
(command.equals("DEPOSIT"))
{
double amount = Double.parseDouble(
tokenizer.nextToken());
bank.deposit(account, amount);
}
else if (command.equals("WITHDRAW"))
{
double amount = Double.parseDouble(
tokenizer.nextToken());
bank.withdraw(account, amount);
}
else if (!command.equals("BALANCE"))
return "Invalid command";
return account + " " + bank.getBalance(account);
}
private Socket s;
private Bank bank;
}
```

File BankClient.java

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;
/**
Questo programma collauda il server bancario.
*/
public class BankClient
{
public static void main(String[] args)
throws IOException
{
final int SBAP_PORT = 8888;
Socket s = new Socket("localhost", SBAP_PORT);
```

```
InputStream in = s.getInputStream();
OutputStream out = s.getOutputStream();
```

```
BufferedReader reader = new BufferedReader(
new InputStreamReader(in));
PrintWriter writer = new PrintWriter(out);
String command = “DEPOSIT 3 1000\n”;
System.out.println(“Sending: “ + command);
writer.print(command);
writer.flush();
String response = reader.readLine();
System.out.println(“Receiving: “ + response);
command = “WITHDRAW 3 500\n”;
System.out.println(“Sending: “ + command);
writer.print(command);
writer.flush();
response = reader.readLine();
System.out.println(“Receiving: “ + response);
command = “QUIT\n”;
System.out.println(“Sending: “ + command);
writer.print(command);
writer.flush();
s.close();
}
}
```

File Bank.java

```
/**
Una banca è composta da più conti bancari.
*/
public class Bank
{
/**
Costruisce una banca con un numero di conti bancari assegnato.
@param size il numero di conti
*/
public Bank(int size)
```

```
{
accounts = new BankAccount[size];
for (int i = 0; i < accounts.length; i++)
accounts[i] = new BankAccount();
}
/**
Versa denaro in un conto bancario.
@param accountNumber il numero di conto
@param amount l'importo da versare
*/
public void deposit(int accountNumber, double amount)
{
BankAccount account = accounts[accountNumber];
account.deposit(amount);
}
```



```
/**
Preleva denaro da un conto bancario.
@param accountNumber il numero di conto
@param amount l'importo da prelevare
*/
public void withdraw(int accountNumber, double amount)
{
BankAccount account = accounts[accountNumber];
account.withdraw(amount);
}
/**
Fornisce il saldo di un conto bancario.
@param accountNumber il numero di conto
@return il saldo del conto
*/
public double getBalance(int accountNumber)
{
BankAccount account = accounts[accountNumber];
return account.getBalance();
}
private BankAccount[] accounts;
```

}

Consigli pratici 1

Progettare programmi client/server

Il server bancario di questo paragrafo è un esempio tipico di programma client/server.

Un browser Web e un server Web costituiscono un altro esempio. Quando progettate un'applicazione client/server, seguite questi passi.

Passo 1. *Determinate se ha veramente senso realizzare un server a sé stante e un corrispondente client*

Molte volte ha più senso costruire, invece, un'applicazione Web. Un successivo Capitolo presenterà in dettaglio la progettazione di applicazioni Web. Ad esempio, l'applicazione bancaria di questo paragrafo potrebbe essere facilmente trasformata in un'applicazione Web, usando un form HTML con i pulsanti Withdraw e Deposit. Al contrario, programmi per le chat oppure per la condivisione di file peer-to-peer non possono essere facilmente realizzati come applicazioni Web.

Passo 2. *Progettate un protocollo di comunicazione Individuate con precisione quali messaggi si scambiano il client e il server e quali sono le risposte inviate in caso di errore o in caso di successo.*

Per ogni richiesta e risposta, chiedetevi come venga indicata la fine dei dati.

- *I dati sono contenuti su un'unica riga? In questo caso la fine della riga funge da terminazione per i dati.*
- *I dati possono essere terminati da una riga speciale (come la riga vuota dopo l'intestazione HTTP o la riga contenente soltanto un punto nel protocollo SMTP)?*
- *Chi invia i dati chiude il socket al termine? Questo è ciò che fa un server Web alla fine della risposta a una richiesta di tipo GET.*
- *Chi invia i dati è in grado di dire quanti byte sono contenuti nella richiesta? I browser Web lo fanno nelle richieste di tipo POST.*

Per la comunicazione tra client e server usate il formato di testo, non il formato binario. Un protocollo di tipo testo è più facile da sottoporre a debugging.

Passo 3. *Realizzate il programma server*

Il server aspetta richieste di connessione attraverso un socket e le accetta. Quindi, riceve comandi, li interpreta, e invia una risposta al client.

Passo 4. *Collaudate il server usando il programma telnet Provate tutti i comandi del protocollo di comunicazione.*

Passo 5. *Quando il server funziona, scrivete un programma client Il programma client interagisce con l'utente, ne trasforma le richieste in comandi del protocollo, invia i comandi al server, riceve la risposta e la visualizza per l'utente.*

5

Connessioni URL

Nel Paragrafo 3 avete visto come usare i socket per connettervi a un server Web e come ottenere informazioni dal server inviando comandi HTTP. Data l'importanza del protocollo HTTP, la libreria Java contiene anche una classe URLConnection, che fornisce un comodo supporto per il protocollo HTTP. La classe URLConnection si occupa della connessione al socket, in modo che non vi dobbiate preoccupare dei socket quando volete recuperare informazioni da un server Web. In più, la classe URLConnection è anche in grado di gestire il protocollo FTP (File Transfer Protocol).

La classe URLConnection ren-

La classe URLConnection rende molto semplice il reperimento di un de agevole la comunicazione con

file da un server Web, a partire da uno URL che rappresenti il file sotto un server Web senza dover informa di stringa. Per prima cosa, costruite un oggetto di tipo URL a par-viare comandi HTTP.

tire da uno URL nel formato che vi è familiare, iniziando con il prefisso http o ftp. Quindi, usate il metodo openConnection() dell'oggetto di tipo URL per ottenere proprio un oggetto di tipo URLConnection.

URL u = new URL("http://java.sun.com/index.html"); URLConnection connection = u.openConnection();

Successivamente, invocate il metodo getInputStream per ottenere un flusso di ingresso:

```
InputStream in = connection.getInputStream();
```

A volte usate il flusso di ingresso soltanto per passarlo a un altro metodo; se, però, volete leggerne il contenuto direttamente, dovete trasformarlo in un lettore.

```
BufferedReader reader = new BufferedReader(
new InputStreamReader(in));
```

A questo punto potete leggere, una riga alla volta, il contenuto della risorsa: boolean done = false;

```
while (!done)
{
String input = reader.readLine();
if (input == null)
done = true;
else fa qualcosa con la riga appena letta
}
```

Le classi URLConnection e Http-

La classe URLConnection può fornire ulteriori utili informazioni. Per com-URLConnection vi forniscono

prendere queste possibilità, dovete dare un'occhiata più approfondita ulteriori informazioni sulle richie-alle richieste e risposte HTTP. Avete visto nel Paragrafo 2 che il coman-ste e risposte HTTP.

do per ottenere una risorsa da un server è

```
GET risorsa HTTP/1.0
riga vuota
```

Può darsi che vi siate chiesti per quale strano motivo ci sia bisogno di una riga vuota.

Questa riga vuota fa parte del formato più generico della richiesta HTTP. La prima riga della richiesta è un comando, come GET o POST. Il comando può essere seguito da proprietà della richiesta, e alcuni comandi (in particolare, il comando POST) inviano dati in ingresso al server. (Vedremo il comando POST nel prossimo paragrafo.) La riga vuota serve, quindi, a delimitare il confine tra il comando, con la sua sezione

contenente le proprietà della richiesta, e la sezione con i dati da inviare.

Una tipica proprietà della richiesta è If-Modified-Since. Se richiedete una risorsa in questo modo:

GET risorsa HTTP/1.0

If-Modified-Since: data e ora

riga vuota

il server invia la risorsa soltanto se è più recente dell'ora e della data indicate. I browser usano questa proprietà per velocizzare la visualizzazione di pagine Web già caricate in precedenza. Quando una pagina Web viene caricata, il browser la memorizza in una cartella cache; se in seguito l'utente vuol rivedere la stessa pagina Web, il browser chiede al server di inviare una nuova pagina soltanto se è stata modificata dopo la

data e l'ora in cui aveva inserito la copia nella cache. Se non è così, il browser semplicemente visualizza di nuovo la copia presente nella cache, senza stare a perder tempo per scaricarne un'altra copia identica.

La classe URLConnection ha dei metodi per impostare le proprietà della richiesta.

Ad esempio, potete impostare la proprietà If-Modified-Since invocando il metodo setIfModifiedSince:

connection.setIfModifiedSince(data);

Le proprietà della richiesta vanno impostate prima di chiamare il metodo getInputStream: in questo modo, la classe URLConnection invia al server Web tutte le proprietà della richiesta che sono state impostate.

In modo simile, la risposta fornita dal server inizia con una riga di stato, seguita da un insieme di parametri della risposta, che sono terminati da una riga vuota, a sua volta seguita dai dati richiesti (ad esempio, una pagina HTML). Ecco una tipica risposta:

HTTP/1.1 200 OK

Date: Sun, 29 Aug 1999 00:15:48 GMT

Server: Apache/1.3.3 (Unix)

Last-Modified: Thu, 24 Jun 1999 20:53:38 GMT

Content-Length: 4813
Content-Type: text/html
riga vuota
dati richiesti

Normalmente, il codice della risposta non viene visualizzato, ma può darsi che vi sia capitato di seguire collegamenti errati e di vedere una pagina contenente il codice di risposta 404 Not Found. (La risposta a una richiesta corretta ha invece lo stato 200 OK.) Per esaminare il codice della risposta, dovete eseguire un cast sull'oggetto di tipo URLConnection, forzandolo a diventare un riferimento alla sottoclasse HttpURLConnection. Con i metodi getResponseCode e getResponseMessage potete esaminare il codice della risposta (come il numero 200 in questo esempio, oppure il codice 404 se la pagina non è stata trovata) e il messaggio della risposta: HttpURLConnection httpConnection =

(HttpURLConnection)connection;
int code = httpConnection.getResponseCode(); // es. 404
String message =
httpConnection.getResponseMessage(); // es. "Not Found"

Come potete vedere dalla risposta fornita come esempio, il server invia alcune informazioni a proposito dei dati richiesti, come la loro lunghezza e il loro tipo. Potete esaminare queste informazioni con metodi della classe URLConnection: int length = connection.getContentLength();

String type = connection.getContentType();

Questi metodi possono essere invocati soltanto dopo aver invocato il metodo getInputStream.

Per riassumere: non c'è bisogno di usare i socket per comunicare con un server Web, e non è necessario imparare i dettagli del protocollo HTTP; usate semplicemente le classi URLConnection e HttpURLConnection per ottenere dati da un server Web, per impostare i parametri della richiesta, o per ottenere informazioni riguardanti la risposta.

Il programma che si trova al termine di questo paragrafo mette all'opera la classe URLConnection. Il programma svolge

gli stessi compiti di quello del Paragrafo 3, cioè recupera una pagina Web da un server, ma opera a un livello di astrazione più elevato.

Non c'è più bisogno di inviare esplicitamente un comando GET: se ne occupa la classe URLConnection. Analogamente, l'analisi delle intestazioni della richiesta e della risposta HTTP viene gestita in modo trasparente per il programmatore, e il nostro programma di esempio si avvantaggia di ciò: verifica che la risposta del server sia 200 OK

e, in caso contrario, stampa la risposta del server e termina la propria esecuzione.

Potete provare questo fatto collaudando il programma con un URL errato, come http://

java.sun.com/wombat.html: in questo caso il programma stampa la risposta del server, 404 Not Found.

Nel paragrafo successivo vedrete un altro uso della classe HttpURLConnection: per inviare dati in ingresso a programmi eseguiti sul server.

File URLGet.java

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
/**
Questo programma mostra come usare una connessione URL
per comunicare con un server Web. Fornite sulla riga di comando uno URL, come nell'esempio seguente:
java URLGet http://java.sun.com/index.html
*/
public class URLGet
{
public static void main(String[] args)
throws IOException
```

```
{
// leggi gli argomenti della riga comandi
if (args.length != 1)
{
System.out.println("usage: java URLGet URL");
System.exit(0);
}
// apri la connessione
```

```
URL u = new URL(args[0]);
URLConnection connection = u.openConnection();
// verifica che il codice della risposta sia HTTP_OK (200)
HttpURLConnection httpConnection =
(HttpURLConnection)connection;
int code = httpConnection.getResponseCode();
if (code != HttpURLConnection.HTTP_OK)
{
String message =
httpConnection.getResponseMessage();
System.out.println(code + " " + message);
return;
}
// legge la risposta del server
InputStream in = s.getInputStream();
BufferedReader reader = new BufferedReader(
new InputStreamReader(in));
boolean done = false;
while (!done)
{
String input = reader.readLine();
if (input == null) done = true;
else System.out.println(input);
}
}
}
```


Inviare i dati di un modulo

I contenuti dinamici per il Web

Quando richiedete una pagina Web, come index.html, il server Web vengono generati da programmi

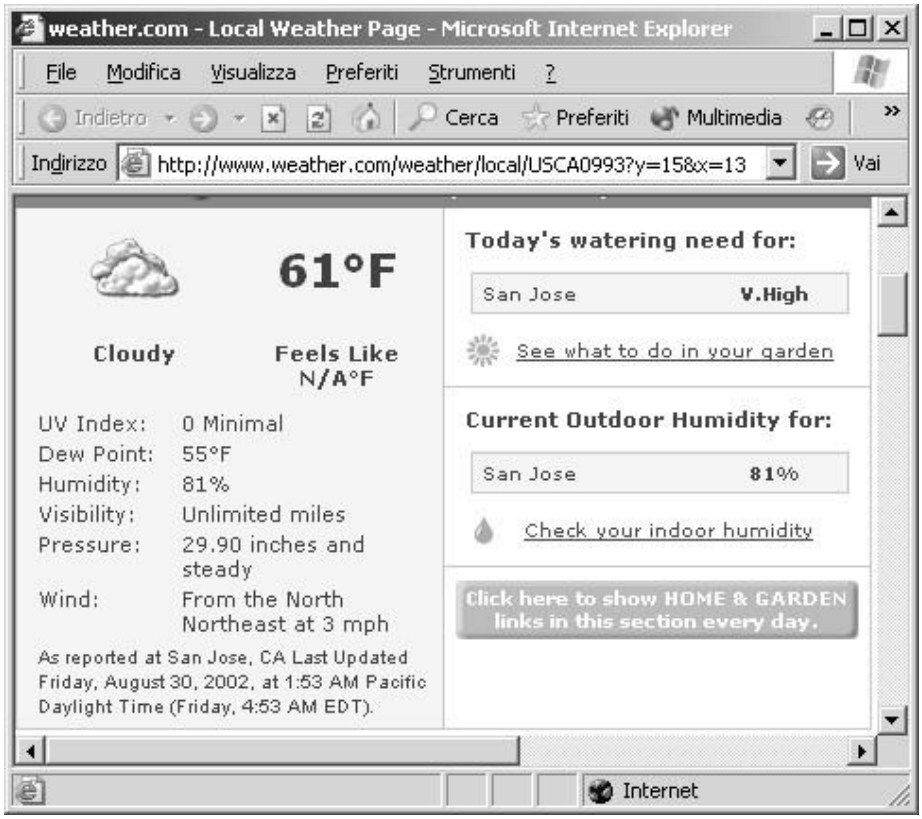
recupera semplicemente il file e ve ne invia il contenuto; ovviamente, se invocati dal server Web quando

richiedete più volte la stessa pagina, riceverete ogni volta gli stessi con-vengono richiesti tali dati.

tenuti, finché qualcuno non sostituirà il file sul server Web. Tuttavia, molte pagine Web hanno un contenuto dinamico: informazioni che vengono continuamente modificate, come le previsioni del tempo o le quotazioni azionarie.

Queste pagine Web non sono memorizzate in file, ma vengono generate da programmi in esecuzione sul server Web: ogni volta che richiedete una di tali pagine, il server Web fa partire il programma appropriato, fornendogli in ingresso i dati che provengo-no dalla richiesta, e restituisce ciò che viene prodotto (osservate la Figura 8). Di solito, questi programmi generano una miscela di dati (come le informazioni meteorologi-che o le quotazioni azionarie che erano state richieste) e di codice HTML per impagi-narli, e il risultato complessivo può essere visualizzato da qualsiasi browser.

Un server Web deve sapere quando uno URL indica un file e quando, invece, indica un programma. Tipicamente, i server Web guardano alla cartella indicata nello URL: ad esempio, tutti gli URL il cui suffisso inizia con /servlet/ oppure con /cgi-bin/



24

Programmazione di rete

Figura 8

Dati meteorologici

dinamici

sono solitamente considerati come riferimenti a programmi, ma i dettagli dipendono dalla configurazione del server Web. Esistono diversi metodi per eseguire programmi sul lato server, come, ad esempio, la tecnologia Java Server Pages. Per ora, assumiamo che il server Web sappia come identificare ed eseguire l'applicazione giusta quando riceve una richiesta con uno URL che indica un programma.

Esistono due comandi HTTP per

Quando il server Web esegue il programma, gli fornisce in ingresso i fornire dati in ingresso a program-dati che ha ricevuto dal client attraverso la richiesta HTTP. Questi dati mi eseguiti sul lato server: GET

in ingresso possono essere inviati in due modi, il metodo GET e il meto-e POST. La sintassi di GET è più

do POST. Con il metodo GET, i dati vengono allegati allo URL che indica semplice, ma può essere utilizzata soltanto per piccole quanti-

il programma, preceduti da un carattere ?. Ad esempio, considerate lo tà di dati.

URL

http://mach.usno.navy.mil/cgi-bin/aa_moonphases?year=2000

Questo URL comunica al server Web mach.usno.navy.mil di eseguire il programma associato a /cgi-bin/aa_moonphases e di fornirgli in ingresso il dato year=2000

Questo metodo per fornire dati in ingresso viene chiamato metodo GET perché usa il comando GET del protocollo HTTP per trasmettere la richiesta al server Web. La richiesta è questa:

GET /cgi-bin/aa_moonphases?year=2000 HTTP/1.0

riga vuota

Solitamente, i programmi eseguiti sul lato server ricevono in ingresso dati costituiti da coppie nome/valore. In questo esempio, il programma riceve un unico dato in ingresso, il cui nome è year e il cui valore è 2000. Più coppie nome/valore vanno separate da un carattere & (ampersand), in questo modo: city=Chicago&zip=60614

Nelle richieste

Se i nomi o i valori contengono caratteri che non siano lettere o numeri GET e POST, i dati

devono essere codificati. A que-

nel codice ASCII, devono essere codificati secondo uno schema detto sto scopo, usate la classe URL-codifica URL. Gli spazi vengono codificati con caratteri +, come in que-Encoder.

sto esempio

city=San+Francisco

Altri byte vengono codificati con % xy, dove xy è il valore esadecimale del byte. Ad esempio, per codificare un carattere / (che ha il codice decimale 47 o esadecimale 2F), usate %2F:

city=Raleigh%2FDurham

Per codificare e decodificare stringhe, potete usare i metodi statici URLEncoder.encode e URLDecoder.decode. Ad esempio

String encoding = URLEncoder.encode(name)

+ “=” + URLEncoder.encode(value);

Il metodo GET ha un grande svantaggio: molti browser pongono un limite al numero di byte ammissibili in uno URL (a volte soltanto 255 caratteri). Quindi, con una richiesta GET potete fornire soltanto una piccola quantità di informazione a un programma eseguito sul lato server. Se volete fornire più informazioni, dovete usare una richiesta di tipo POST, il cui formato è

POST url

Content-Type: tipo dei dati

Content-Length: lunghezza dei dati

altre intestazioni della richiesta

riga vuota

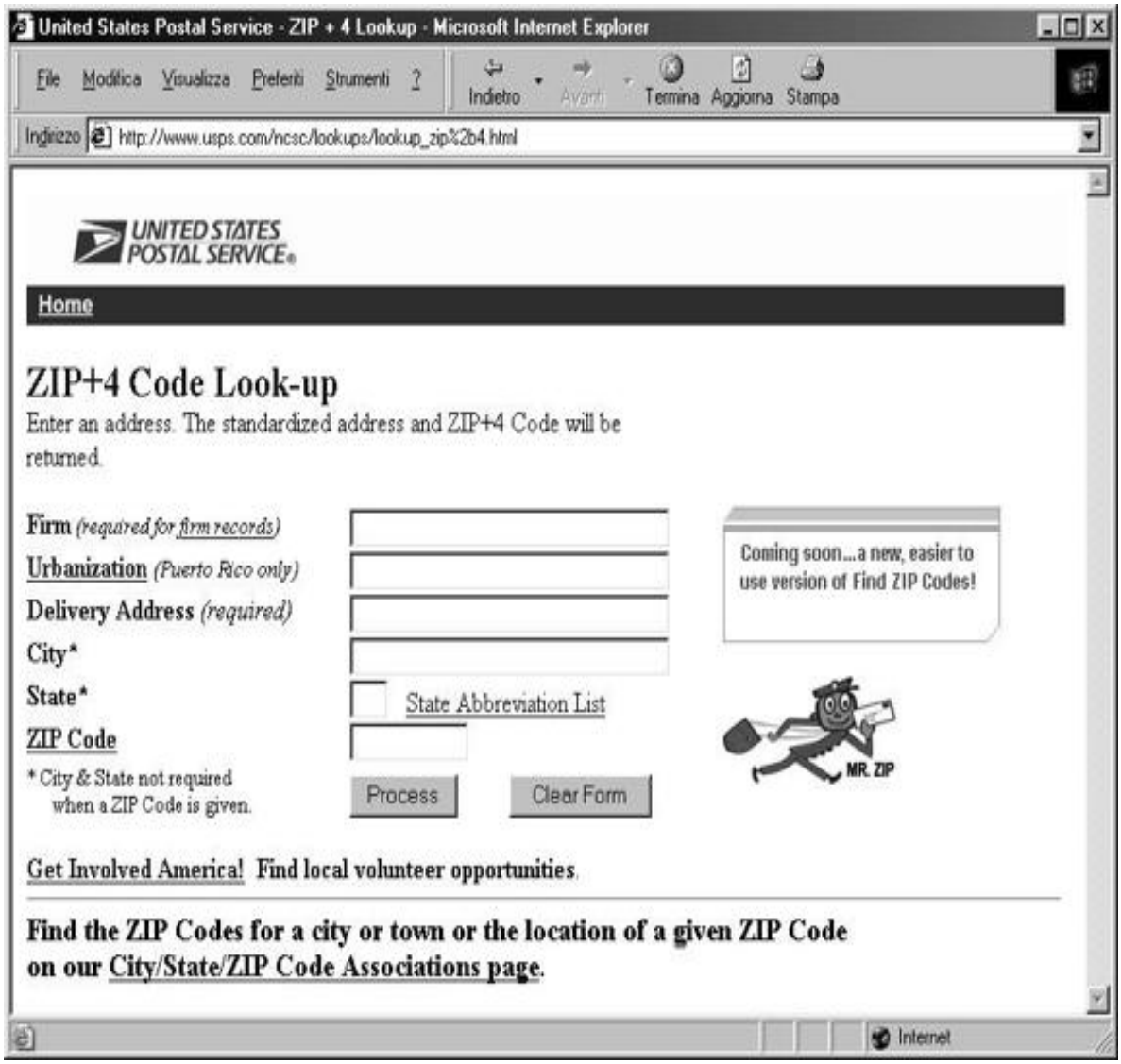
dati in ingresso

I dati in ingresso possono avere qualsiasi formato, ma una frequente fonte di richieste di tipo POST è l’invio di moduli (form) HTML contenenti dati (osservate la Figura 9 per un esempio di form HTML). In questo caso, i browser usano la codifica URL per la trasmissione dei dati, per cui una classica richiesta di tipo POST assomiglia a questa: POST /cgi-bin/zip4/zip4inq2

Content-Type: application/x-www-form-urlencoded

Content-Length: 91

riga vuota



Programmazione di rete

Figura 9

Un form HTML

Firm=&Urbanization=&Delivery+Address=100+Main+Street &City=Ann+

Arbor&State=MI&Zip+Code=48104

Per inviare richieste di tipo POST

Fortunatamente, quando usate la classe URLConnection non vi dovete a un server Web, usate la classe

preoccupare del formato di una richiesta di tipo POST: invocate sempli-URLConnection.

cemente il metodo setDoOutput per avvertire l'oggetto di tipo URLConnection che volete inviare dei dati in uscita, quindi invocate il metodo getOutputStream per ottenere un flusso d'uscita e inviate i dati a tale flusso. Se state inviando dati a un programma sul lato server che elabora i dati di un form, inviate al flusso d'uscita coppie nome/valore nella codifica URL e, quando avete terminato, chiu-dete il flusso. A questo punto, i vostri dati in uscita diventano i dati in ingresso della richiesta POST. Ecco un tipico frammento di codice:

```
connection.setDoOutput(true);
OutputStream out = connection.getOutputStream();
PrintWriter writer = new PrintWriter(out);
writer.print(URLEncoder.encode(name_1)
+ "=" + URLEncoder.encode(value_1) + "&");
...
writer.print(URLEncoder.encode(name_n)
+ "=" + URLEncoder.encode(value_n) + "\n");
writer.close();
```


Il seguente programma di prova effettua la ricerca di un codice postale statunitense (ZIP) invocando un programma sul lato server messo a disposizione dallo United States Postal Service. Tale programma corrisponde allo URL http://www.usps.gov/

cgi-bin/zip4/ctystzip e si aspetta di ricevere un comando di tipo POST avente come dati il solo nome ctystzip associato a un valore che sia un codice ZIP o l'indicazione di una città e di uno stato (come "Beverly Hills, CA").

Per eseguire il programma di prova, fornite i dati in ingresso sulla riga di comando, come

```
java PostZipQuery 12345
```

oppure

```
java PostZipQuery Beverly Hills, CA
```

Il programma, per prima cosa, concatena tutti gli argomenti ricevuti sulla riga di comando per formare la stringa input, oppure imposta input al valore "90210" (il codice ZIP di Beverly

Hills, California) se non fornite alcun argomento sulla riga di comando.

Poi, il programma passa i dati in ingresso all'oggetto di tipo URLConnection, si connette al server del servizio postale e visualizza la risposta ricevuta dal programma eseguito sul lato server. La risposta ha il formato HTML, ma se lo esaminate attentamente potete carpire la risposta. L'esercizio P.6 vi chiede di ripulire ciò che viene visualizzato.

Questo programma completa la nostra introduzione alla programmazione di rete in Java. Avete visto come usare i socket per connettere fra loro programmi client e programmi server. Avete anche visto come usare la classe URLConnection, di più alto livello, per ottenere informazioni da un server Web. Tale classe è più semplice da usare, perché si fa carico dei dettagli relativi alla connessione dei socket e implementa il protocollo HTTP.

File PostZipQuery.java

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;
/**
```

Questo programma invia una richiesta al server dello United States Postal Service per cercare il codice ZIP a partire dal nome di una città. Fornite sulla riga di comando il nome della città e il nome dello stato (facoltativo), come nell'esempio seguente:

```
java PostZipQuery Los Angeles, CA
*/
```



```
public class PostZipQuery
{
```

```
public static void main(String[] args)
throws IOException
{
// concatena tutti gli argomenti della riga di comando String input;
if (args.length > 0)
{
input = args[0];
for (int i = 1; i < args.length; i++)
input += “ “ + args[i];
}
else
input = “90210”;
// stabilisci una connessione URL
URL u = new URL(
“http://www.usps.gov/cgi-bin/zip4/ctystzip”);
URLConnection connection = u.openConnection();
// invia i dati del comando POST
connection.setDoOutput(true);
OutputStream out = connection.getOutputStream();
PrintWriter writer = new PrintWriter(out);
writer.print(“ctystzip=”
+ URLEncoder.encode(input) + “\n”);
writer.close();
// visualizza la risposta ricevuta dal server
InputStream in = connection.getInputStream();
BufferedReader reader = new BufferedReader(
new InputStreamReader(in));
boolean done = false;
while (!done)
{
String inputLine = reader.readLine();
if (inputLine == null)
done = true;
else
System.out.println(inputLine);
}
reader.close();
```

}
}

Consigli per la produttività 1

Usare librerie di alto livello

Quando comunicate con un server Web per ottenere dei dati, avete due scelte. Potete creare una connessione di socket e inviare comandi GET e POST al server attraverso il socket, oppure potete usare la classe URLConnection e lasciare che essa invii i comandi da parte vostra.

Analogamente, per comunicare con un server di posta elettronica, potete scrivere programmi che inviano comandi SMTP e POP, oppure potete imparare a utilizzare le estensioni di Java per la posta elettronica (consultate http://java.sun.com/products/

javamail/index.html per avere maggiori informazioni sulla libreria JavaMail).

In tale situazione, potreste avere la tentazione di usare l'approccio di basso livello e di inviare i comandi attraverso una connessione di socket, perché sembra più semplice dell'apprendimento di un complicato insieme di classi, ma tale semplicità è spesso fuorviante. Quando andate al di là dei casi più semplici, l'approccio di basso livello solitamente richiede molta applicazione. Ad esempio, per inviare messaggi di posta elettronica con allegati in formato binario, dovreste imparare complesse codifiche dei dati. Le librerie di alto livello hanno al loro interno tutte queste conoscenze, in modo che non dobbiate inventare nuovamente la ruota.

Per tale motivo, non dovreste davvero usare i socket per connettervi a server Web: usate, invece, sempre la classe URLConnection. Perché vi abbiamo insegnato i socket se non è previsto che li utilizziate? Ci sono due motivi. Alcuni programmi di tipo client non comunicano con server Web o server di posta elettronica, e può darsi che dobbiate usare i socket quando non è disponibile una libreria di alto livello. Inoltre, è altrettanto importante sapere ciò che in realtà viene fatto da una libreria di alto livello, perché vi aiuta a capire meglio il suo funzionamento.

Per la stessa ragione avete visto come realizzare liste concatenate, anche se probabilmente non realizzerete mai vostre liste e userete soltanto la classe LinkedList.

Riepilogo del capitolo

1. Internet è un insieme di reti mondiali, di apparecchiature di instradamento e di computer che usano un insieme comune di protocolli per definire come ciascuna parte della rete interagisce con le altre componenti.

2. TCP/IP è l'abbreviazione di Transmission Control Protocol su Internet Protocol, la coppia di protocolli di comunicazione usata per stabilire una trasmissione affidabile di dati fra due computer di Internet.

3. Una connessione TCP richiede gli indirizzi Internet e i numeri di porta di entrambe le sue terminazioni.

4. HTTP (Hypertext Transfer Protocol) è il protocollo che definisce la comunicazione fra browser Web e server Web.

5. Un URL (Uniform Resource Locator) è un riferimento a una risorsa informativa (come una pagina Web o un'immagine) che si trova nel World Wide Web.

6. Il programma telnet è uno strumento utile per stabilire connessioni di prova con i server.

7. Il comando HTTP GET richiede informazioni a un server Web, il quale restituisce la risorsa richiesta, che può essere una pagina Web, un'immagine, o altri dati.

8. Un socket è un oggetto che incapsula una connessione TCP/IP. Per comunicare con l'altra estremità della connessione, usate i flussi di ingresso e uscita connessi al socket.

9. Quando la trasmissione attraverso un socket è terminata, ricordatevi di chiudere il socket stesso.

10. In protocolli di tipo testo, trasformate i flussi dei socket in lettori e scrittori.

11. Alla fine di ogni comando svuotate il buffer dello scrittore connesso a un socket.

In questo modo il comando viene effettivamente inviato al server, anche se il buffer non è pieno.

12. La classe ServerSocket viene usata dalle applicazioni server per restare in attesa di connessioni da parte dei client.

13. La classe URLConnection rende agevole la comunicazione con un server Web senza dover inviare comandi HTTP.

14. Le classi URLConnection e HttpURLConnection vi forniscono ulteriori informazioni sulle richieste e risposte HTTP.

15. I contenuti dinamici per il Web vengono generati da programmi invocati dal server Web quando vengono richiesti tali dati.

16. Esistono due comandi HTTP per fornire dati in ingresso a programmi eseguiti sul lato server: GET e POST. La sintassi di GET è più semplice, ma può essere utilizzata soltanto per piccole quantità di dati.

17. Nelle richieste GET e POST, i dati devono essere codificati. A questo scopo, usate la classe URLEncoder.

18. Per inviare richieste di tipo POST a un server Web, usate la classe URLConnection.

Classi, oggetti e metodi presentati nel capitolo

java.net.ServerSocket
accept
close
java.net.Socket
getInputStream
getOutputStream
java.net.URL
openConnection
java.net.URLConnection
getInputStream

setIfModifiedSince
getContentLength
getContentType
getOutputStream
setDoOutput
java.net.HttpURLConnection

getResponseCode
getResponseMessage
java.net.URLDecoder
decode
java.net.URLEncoder
encode

Esercizi di ripasso

Esercizio R.1. *Cos'è un server? Cos'è un client? Quanti client possono connettersi contemporaneamente a un server?*

Esercizio R.2. *Cos'è un socket? Qual è la differenza tra un oggetto di tipo Socket e un oggetto di tipo ServerSocket?*

Esercizio R.3. *In quali circostanze viene lanciata un'eccezione di tipo UnknownHostException?*

Esercizio R.4. *Cosa succede se il secondo parametro del costruttore di Socket non è uguale al numero della porta che il server sta utilizzando per attendere connessioni?*

Esercizio R.5. *Quando viene creato un socket, quale indirizzo Internet viene usato?*

- *L'indirizzo del computer al quale ci si vuole connettere*
- *L'indirizzo del vostro computer*
- *L'indirizzo del vostro ISP*

Esercizio R.6. *Qual è lo scopo del metodo accept della classe ServerSocket?*

Esercizio R.7. *Dopo che un socket ha stabilito una connessione, quale meccanismo userà il vostro programma client per leggere i dati ricevuti dal server?*

- *L'oggetto di tipo Socket riempirà un buffer di byte.*
- *Userete un oggetto di tipo Reader ottenuto dall'oggetto di tipo Socket*
- *Userete un oggetto di tipo InputStream ottenuto dall'oggetto di tipo Socket* ***Esercizio R.8.*** *Perché di solito non si opera direttamente con gli oggetti di tipo InputStream e OutputStream ottenuti da un oggetto di tipo Socket?*

Esercizio R.9. *Quando un programma client comunica con un server, a volte ha bisogno di svuotare il buffer del flusso di uscita. Spiegate perché.*

Esercizio R.10. *Nell'esempio della classe BankService, qual è l'utilità della classe StringTokenizer?*

Esercizio R.11. *Come si può comunicare con un server Web senza usare i socket?*

Esercizio R.12. *Qual è la differenza tra HTTP e HTML?*

Esercizio R.13. *Cos'è uno URL? Come si crea uno URL? Come ci si connette a uno URL?*

Esercizio R.14. *Qual è la differenza tra un oggetto di tipo URL e un oggetto di tipo URLConnection?*

Esercizio R.15. *Qual è la differenza tra il metodo GET e il metodo POST?*

Esercizio R.16. *Vero o falso?*

■ *La classe ServerSocket viene usata soltanto dai programmi client.*

■ *La classe Socket viene usata soltanto dai programmi server.*

■ *Un server Web sa quando uno URL indica un file e quando, invece, indica un programma.*

■ *Mediante una richiesta di tipo GET, potete fornire a un programma eseguito sul lato server tutte le informazioni che volete.*

■ *Il costruttore della classe URL lancia una MalformedURLException quando uno URL*

non viene specificato nel modo corretto.

Esercizi di programmazione

Esercizio P.1. *Modificate il programma WebGet in modo che stampi la sola intestazione HTTP della pagina HTML ricevuta. L'intestazione HTTP è l'inizio dei dati contenuti nella risposta e consiste di righe come le seguenti: HTTP/1.1 200 OK*

Date: Sun, 10 Jun 2001 16:10:34 GMT

Server: Apache/1.3.19 (Unix)

Cache-Control: max-age=86400

Expires: Mon, 11 Jun 2001 16:10:34 GMT

Connection: close

Content-Type: text/html

seguite da una riga vuota.

Esercizio P.2. *Modificate il programma WebGet in modo che stampi soltanto il titolo della pagina HTML ricevuta. Una pagina*

HTML ha la struttura seguente:

<html><head><title>…</title></head><body>…</body> </html>

Ad esempio, se eseguite il programma digitando sulla riga dei comandi java WebGet java.sun.com index.html, dovrebbe essere visualizzato java.sun.com – The source for Java(TM) Technology.

Esercizio P.3. *Riscrivete il programma PostZipQuery in modo da non usare un oggetto di tipo URLConnection, generando manualmente i dati per la richiesta di tipo POST e inviandola al server.*

Esercizio P.4. *Modificate il programma BankServer fornendo un controllo completo degli errori. Ad esempio, una cosa da controllare a ogni prelievo è che ci sia denaro a sufficienza nel conto. Inviate al client i messaggi di errore appropriati. Migliorate il protocollo in modo che sia simile al protocollo HTTP, nel quale ciascuna risposta del server inizia con un numero che indica una condizione di successo o di fallimento, seguito da una stringa contenente i dati della risposta o una descrizione dell'errore.*

Esercizio P.5. *Modificate il programma BankServer in modo che accetti connessioni simultanee da più client. Dovete apportare le seguenti modifiche:*

- *La classe BankService deve estendere Thread. Rinominate il metodo doService in run.*
- *Quando il socket di tipo server accetta una connessione, deve costruire un nuovo thread di tipo BankService e farlo partire.*
- *I metodi della classe BankAccount devono essere sincronizzati.*

Provate il programma stabilendo due connessioni telnet simultanee con il server.

Esercizio P.6. *Modificate il programma PostZipQuery ripulendo il risultato prodotto in uscita in modo che la risposta dell'interrogazione sia chiaramente visibile. Suggerimento: Realizzate una classe che converta il codice HTML in puro testo*

eliminando i marcatori indesiderati e traducendo i marcatori <br> e <p> in caratteri '\n'.

Esercizio P.7. *Scrivete un'applicazione client che esegua un ciclo infinito con le seguenti azioni: (a) chiede un numero all'utente, (b) invia il valore al server, (c) riceve un numero, (d) visualizza il numero ricevuto. Scrivete anche un server che esegua un ciclo infinito per leggere un numero da un client, calcolarne la radice quadrata e inviare il risultato al client.*

Esercizio P.8. *Realizzate un programma client/server in cui il client visualizza l'ora e la data fornite dal server. Si devono implementare due classi: DateClient e DateServer. La classe DateServer invia semplicemente la stringa new Date().toString() ogni volta che accetta una connessione, poi chiude il socket.*

Esercizio P.9. *Scrivete un programma che visualizzi il protocollo, il nome del computer host, la porta e il file che compongono uno URL. Suggerimento: Consultate la documentazione della classe URL di libreria.*

Esercizio P.10. *Scrivete un semplice server Web che riconosca soltanto la richiesta di tipo GET. Quando un client si connette al vostro server e invia un comando come GET*

nomeFile HTTP/1.0, restituite l'intestazione

HTTP/1.1 200 OK

seguita da una riga vuota e da tutte le righe che costituiscono il file. Invece, se il file non esiste, restituite 404 Not Found.

Collaudate il vostro server Web eseguendo il vostro browser Web e caricando una pagina come, ad esempio, localhost:/c:\cs1\myfile.html.

Esercizio P.11. *Scrivete un programma client/server per un servizio di chat. Il server di chat accetta connessioni dai client. Ogni volta che un client invia un messaggio, esso viene visualizzato da tutti gli altri client. Usate un protocollo con tre comandi: LOGIN*

nome, CHAT messaggio, LOGOUT. Avete bisogno di thread separati perché questo programma funzioni. Seguite l'approccio dell'Esercizio P.5.

Esercizio P.12. *Una interrogazione del tipo http://mach.usno.navy.mil/cgi-bin/aa_moonphases?year=2001*

restituisce una pagina che contiene le fasi lunari di un certo anno. Scrivete un programma che chieda all'utente anno, mese e giorno, e che visualizzi la fase lunare di tale giorno.

Connessione a base di dati

Obiettivi del capitolo

- *Capire come vengono memorizzate le informazioni nelle basi di dati relazionali*
- *Imparare a interrogare una base di dati con il linguaggio SQL (Structured Query Language)*
- *Connettersi a basi di dati mediante la tecnologia JDBC (Java Database Connectivity)*
- *Scrivere programmi per basi di dati che inseriscano, aggiornino e interroghino dati in una base di dati relazionale*

2

Connessione a base di dati

Una base di dati (database) rela-

Quando si memorizzano dati, si vuole essere in grado di aggiungerne zionale memorizza le informazioni

altri, di rimuoverne altri ancora, di modificarli e di cercare dati che sod-all'interno di tabelle. Ciascuna co-

disfino particolari criteri. Tuttavia, se avete a che fare con molti dati, può lonna di una tabella è caratteriz-essere difficile eseguire questi compiti in modo veloce ed efficiente. Dato zata da un nome e da un tipo di

dati.

che la memorizzazione dei dati è un compito così comune, sono stati inventati speciali sistemi per la gestione delle basi di dati (DBMS, Database Management System) che consentono al programmatore di ragionare sui dati anziché sulla loro memorizzazione nei file. I DBMS vengono usati molto comunemente per conservare dati sui clienti, informazioni mediche, e molti altri tipi di dati. Il tipo di basi di dati più comune è quello relazionale: in una base di dati di questo tipo, potete usare un linguaggio di interrogazione strutturato per formulare interrogazioni e istruzioni di aggiornamento in un modo naturale, senza dovervi preoccupare di dove e come

l'informazione sia memorizzata. In questo capitolo imparerete come interrogare e aggiornare le informazioni all'interno di una base di dati relazionale e come accedere da un programma Java alle informazioni contenute in una base di dati.

Ovviamente, la gestione delle basi di dati è un argomento complesso e vi sono molti aspetti di grande importanza nella realizzazione di sistemi di basi di dati professionali. Questo capitolo termina con una breve discussione di alcuni degli argomenti più avanzati che riguardano le basi di dati.

1 Organizzare le informazioni in una base di dati

1.1 Tabelle delle basi di dati

Una base di dati (database) relazionale memorizza le informazioni in tabelle, di cui la Figura 1 è un tipico esempio. Come potete vedere, ciascuna riga di questa tabella corrisponde a un prodotto. Le intestazioni delle colonne corrispondono ad attributi del prodotto: il codice prodotto, la descrizione e il prezzo unitario. Notate che tutti i dati di una colonna sono dello stesso tipo: i codici prodotto e le descrizioni sono stringhe, mentre i prezzi unitari sono numeri in virgola mobile. I tipi di colonne disponibili possono essere un po' diversi da una base di dati a un'altra: la Tabella 1 mostra i tipi più comunemente disponibili nelle basi di dati relazionali che aderiscono allo standard SQL (Structured Query Language, spesso pronunciato come la parola inglese

"sequel"). (Si veda la citazione [1] per avere maggiori informazioni sullo standard SQL.)

Codice Prodotto
Descrizione
Prezzo Unitario
Figura 1
116-064
Toaster
24.95
Una tabella che descrive prodotti

257-535
Hair dryer
29.95
in una base di dati
643-119
Car vacuum
19.99
relazionale
Connessione a base di dati
3
Tabella 1 *Alcuni tipi di dati dello standard SQL e i loro tipi corrispondenti in Java* ***Tipo di dati SQL***
Tipo di dati Java
INTEGER o INT
int
REAL
float
DOUBLE
double
DECIMAL(m, n)
Numeri decimali in virgola fissa con m cifre in totale ed n cifre dopo il punto decimale. Simile a BigDecimal.
BOOLEAN
boolean
CHARACTER(n) o CHAR(n)
Stringa di lunghezza fissa n. Simile a String.
Il linguaggio SQL (Structured
La maggior parte delle basi di dati relazionali aderiscono allo standard Query Language) è un linguag-SQL. Nel prossimo paragrafo vedrete come usare i comandi SQL per gio imperativo per interagire con
effettuare interrogazioni, ma esistono altri comandi SQL oltre a quelli una base di dati.
per fare interrogazioni. Non c'è alcuna relazione tra SQL e Java: sono linguaggi diversi. Tuttavia, come vedrete più avanti in questo capitolo, potete usare Java per inviare comandi SQL a una base di dati.

Ad esempio, ecco il comando SQL per creare una tabella di prodotti: CREATE TABLE Products

(

Product_Code CHAR(11),

Description CHAR(40),

Unit_Price DECIMAL(10, 2)

)

Diversamente da Java, il linguaggio SQL non è sensibile alla differenza tra lettere maiuscole e minuscole. Ad esempio, avreste potuto scrivere il comando create table invece di CREATE TABLE. Tuttavia, per convenzione, useremo le lettere maiuscole per le parole chiave SQL e parole miste (maiuscole e minuscole) per i nomi di Usate i comandi SQL CREATE

tabelle e colonne.

TABLE e INSERT per aggiunge-

Per inserire righe nella tabella, usate il comando INSERT, fornendo re dati a una base di dati.

un comando per ciascuna riga, in questo modo:

INSERT INTO Products

VALUES (‘257-535’, ‘Hair dryer’, 29.95)

Come potete vedere, il linguaggio SQL usa gli apici singoli, e non i doppi apici, per delimitate le stringhe. Come si risolve il problema di una stringa che contiene un apice singolo? Invece di usare un carattere di escape come in Java (\’), scrivete semplicemente il singolo apice due volte, in questo modo:

‘Sam’’s Small Appliances’

Se create una tabella e successivamente la volete eliminare, usate il comando DROP

TABLE. Ad esempio

DROP TABLE Test

4

Connessione a base di dati

Consigli per la produttività 1

Aderite agli standard

Il linguaggio Java è assai standardizzato: difficilmente troverete compilatori che vi consentano di usare codice Java diverso da quello standard e, se lo fanno, si tratta sempre di un errore del compilatore. Al contrario, le varie implementazioni

del linguaggio SQL sono spesso molto più permissive. Ad esempio, molti produttori di SQL

consentono, all'interno di una stringa SQL, l'utilizzo di sequenze di escape simili a quelle di Java, in questo modo:

'Sam\'s Small Appliances'

Probabilmente il produttore avrà pensato che questo sarebbe stato "utile" ai programmatori che avessero una buona familiarità con i linguaggi Java o C (il linguaggio C usa lo stesso meccanismo di escape per indicare caratteri speciali).

Tuttavia, questa è soltanto un'illusione. Se deviate dallo standard, limitate la portabilità. Supponete di voler successivamente trasferire il codice della vostra base di dati inserendola in un DBMS di un diverso produttore, magari per migliorarne le prestazioni o per ridurre il costo del software di gestione stesso. Se tale altro produttore non ha implementato quella particolare deviazione dallo standard, il vostro codice non funzionerà più e dovrete perdere tempo per sistemarlo.

Per evitare questi problemi, dovreste seguire alla lettera gli standard. Nel caso di SQL, non potete affidarvi alla base di dati per la segnalazione degli errori, perché alcuni di essi potrebbero essere stati considerati "utili" estensioni. Ciò significa che dovete conoscere lo standard e imporvi di seguirlo.

1.2

Collegamenti tra tabelle

Se avete oggetti le cui variabili istanza sono stringhe, numeri, date o altri tipi di dati che siano ammessi come tipo di colonna, allora li potete facilmente memorizzare come righe di una tabella di una base di dati. Ad esempio, considerate una classe Customer come la seguente:

```
class Customer
{
...
private String name;
private String address;
private String city;
private String state;
private String zip;
}
```

In questo caso è semplice ottenere la struttura di una tabella che vi consenta di memorizzare clienti: osservate la Figura 2.

Connessione a base di dati

5

Name

Address

City

State

Zip

Figura 2

CHAR (40)

CHAR (40)

CHAR (30)

CHAR (2)

CHAR (10)

Una tabella per

Sam's Small Appliances 100 Main Street

Anytown

CA

98765

memorizzare clienti

Per oggetti diversi, può non essere così semplice. Considerate una fattura: ciascun oggetto di tipo fattura contiene un riferimento a un oggetto di tipo cliente.

```
class Invoice
{
...
private int invoiceNumber;
private Customer theCustomer;
}
```

Non potete semplicemente predisporre una colonna che contenga un cliente, perché Customer non è un tipo di dati del linguaggio SQL. In questo senso, i campi di una tabella sono molto diversi dalle variabili Java. I tipi di dati possibili per un campo di una tabella sono soltanto i tipi di dati forniti dal linguaggio SQL.

Ovviamente, potreste pensare di inserire semplicemente tutti i dati del cliente nella tabella delle fatture, come vedete

nella Figura 3, ma questa non è una buona idea.

Se osservate i dati forniti come esempio nella Figura 3, noterete che la ditta Sam's Small Appliances ha due fatture, aventi i numeri 11731 e 11733; nonostante ciò, tutte le informazioni del cliente sono replicate in due righe.

Questa duplicazione ha due inconvenienti. Prima di tutto, memorizzare più volte la stessa informazione è uno spreco. Se lo stesso cliente effettua molti ordini, l'informazione replicata può richiedere un sacco di spazio. Ancora più importante, replicare le informazioni è pericoloso. Supponete che il cliente cambi indirizzo: sarebbe un facile errore aggiornare le informazioni del cliente in alcune delle righe che contengono fatture, lasciando il vecchio indirizzo in altre.

In un programma Java, non avreste nessuno dei due problemi: più Dovreste evitare righe con dati

replicati. Distribuite, invece, i dati

oggetti di tipo Invoice possono avere un riferimento a un unico oggetto su più tabelle.

condiviso di tipo Customer.

*Il primo passo per ottenere lo stesso risultato in una base di dati consiste nell'organizzare i dati in più tabelle, come nella Figura 4. Suddividere le co-Invoice_*

Customer_
Customer_
Customer_
Customer_
Customer_
…
Number
Name
Address
City
State
Zip
INTEGER
CHAR (40)
CHAR (40)
CHAR (30)

CHAR (2)
CHAR (10)
...
11731
Sam’s Small
100 Main
Anytown
CA
98765
...
Figura 3
Appliances
Street
Brutto progetto
11732
Electronics
1175 Liberty Pleasantville
MI
45066
...
di una tabella
Unlimited
Ave
per fatture avente
11733
Sam’s Small
100 Main
Anytown
CA
98765
...
i dati dei clienti
Appliances
Street
replicati
6
Connessione a base di dati
Invoice

Invoice_
Customer_
Payment
Number
Number
INTEGER
INTEGER
DECIMAL (10, 2)
11731
3175
0
11732
3176
249.95
11733
3175
0
Customer
Customer_
Name
Address
City
State
Zip
Number
INTEGER
CHAR (40)
CHAR (40)
CHAR (30)
CHAR (2)
CHAR (10)
3175
Sam’s Small
100 Main
Anytown
CA
98765
Figura 4

Appliances
Street
Due tabelle per i dati
3176
Electronics
1175 Liberty Pleasantville
MI
45066
delle fatture
Unlimited
Ave
e dei clienti

lonne in due tabelle risolve il problema della replica dei dati: i dati del cliente non vengono più replicati, perché la tabella delle fatture non contiene informazioni sui clienti e la tabella dei clienti contiene un'unica riga per ciascun cliente. Ma come facciamo a riferirci al cliente al quale è stata emessa una fattura? Notate nella Figura 4

che ora c'è una colonna Customer_Number in entrambe le tabelle Customer e Invoice: tutte le fatture di Sam's Small Appliances condividono soltanto il codice cliente. Le due tabelle sono collegate dal campo Customer_Number. Per avere maggiori dettagli su questo cliente, dovete usare il suo codice cliente e cercarlo nella tabella dei clienti.

Una chiave primaria è una colonna

Notate che il codice cliente è un identificativo univoco: abbiamo in-

(o una combinazione di colonne)

trodotto il codice cliente perché il nome del cliente di per sé potrebbe i cui valori identificano univoca-non essere univoco. Ad esempio, ci possono essere diversi negozi chia-mente una riga di una tabella.

mati Electronics Unlimited in diverse città, quindi il solo nome del cliente non identifica univocamente una riga nella tabella, per cui non possiamo usare il nome come collegamento tra le due tabelle.

Usando la terminologia delle basi di dati, una colonna (o una combinazione di colonne) che identifica univocamente una riga

in una tabella viene detta chiave primaria. Nella nostra tabella Customer, la colonna Customer_Number è una chiave primaria. Non tutte le tabelle delle basi di dati devono avere una chiave primaria: vi serve una chiave primaria soltanto se volete stabilire un collegamento da un'altra tabella.

Ad esempio, la tabella Customer deve avere una chiave primaria, in modo da poter collegare i clienti alle fatture.

Una chiave esterna è un riferimen-

Quando una chiave primaria viene collegata a un'altra tabella, la to a una chiave primaria in una

colonna corrispondente (o la combinazione di colonne) in tale tabella tabella collegata.

viene detta chiave esterna. Ad esempio, Customer_Number nella tabella Invoice è una chiave esterna, collegata alla chiave primaria della tabella Customer. Diversamente dalle chiavi primarie, le chiavi esterne non devono necessariamente essere univoche. Ad esempio, nella tabella Invoice abbiamo diverse righe con lo stesso valore per la chiave esterna Customer_Number.

Connessione a base di dati

7

Consigli per la produttività 2

Evitate le repliche di dati non necessarie

Per i progettisti di basi di dati inesperti, è molto comune replicare dati in una tabella come Invoice. Ogni volta che vi trovate a replicare dei dati in una tabella, chiedetevi se potete spostare i dati replicati in una tabella separata e usare una chiave come un codice o un numero identificativo per collegare le tabelle.

Considerate questo esempio:

...
Product_Code
Product_Description
Product_Price
...
...
CHAR (10)
CHAR (20)

DECIMAL (10, 2)
...
...
116-064
Toaster
24.95
...
...
116-064
Toaster
24.95
...
...
...
...
...
...

Come potete vedere, alcune informazioni sui prodotti sono replicate: si tratta di un errore? Dipende: la descrizione del prodotto avente il codice 116-064 rimarrà sempre

"Toaster", per cui tale corrispondenza dovrebbe essere memorizzata in una tabella Product esterna.

Il prezzo del prodotto, tuttavia, può cambiare nel tempo. Quando ciò accade, le vecchie fatture non devono usare automaticamente il nuovo prezzo, per cui ha senso memorizzare nella tabella Invoice il prezzo realmente addebitato al cliente. L'elenco dei prezzi attuali, però, dovrebbe essere memorizzato in una tabella Product esterna.

1.3
Realizzare relazioni "uno a molti"

Ciascuna fattura è collegata a un solo cliente: si tratta di una relazione uno a uno.

D'altra parte, ciascuna fattura ha più articoli, ognuno dei quali identifica un prodotto venduto e la relativa quantità e prezzo unitario. Quindi, esiste una relazione di tipo uno a molti fra le fatture e gli articoli. Nella classe Java, gli oggetti di tipo Item vengono memorizzati in un vettore:

class

{
...
private int invoiceNumber;
private Customer theCustomer;
private ArrayList items; // contiene oggetti di tipo Item private double payment;
}
Tuttavia, in una base di dati relazionale, dovete memorizzare le informazioni in tabelle; sorprendentemente, molti programmatori, trovandosi di fronte questa situazione, ***8***
Connessione a base di dati
Invoice_
Customer_
Product_
Quantity1
Product_
Quantity2
Product_
Quantity3
Payment
Number
Number
Code1
Code2
Code3
INTEGER
INTEGER
CHAR (10)
INTEGER
CHAR (10)
INTEGER
CHAR (10)
INTEGER
DECIMAL (10, 2)
11731
3175
116-064
3

257-535
1
643-119
2
0
Figura 5
commettono un grave errore e replicano un gruppo di colonne, uno per ciascun arti-Brutto progetto
colo, come nella Figura 5.
di una tabella
Questo progetto è, chiaramente, non soddisfacente. Cosa dovremmo fare nel caso per fatture avente
in cui avessimo una fattura con più di tre articoli? Dovremmo forse avere dieci gruppi colonne replicate
di colonne? Ma questo sarebbe uno spreco, se la maggior parte delle fatture avesse soltanto un paio di articoli, e, comunque, non risolverebbe il problema, nel caso in cui occasionalmente ci fossero fatture con molti articoli.
Invece di fare in questo modo, dovreste nuovamente distribuire l'informazione in due tabelle: una per le fatture e un'altra per gli articoli; collegate, poi, ciascun articolo alla sua fattura con una chiave esterna Invoice_Number nella tabella degli articoli, come potete vedere nella Figura 6.
A questo punto, la nostra base di dati è costituita da quattro tabelle:
- *Invoice*
- *Customer*
- *Item*
- *Product*

La Figura 7 mostra i collegamenti tra queste tabelle. Nel prossimo paragrafo vedrete come interrogare questa base di dati per avere informazioni su fatture, clienti e prodotti: le interrogazioni sfrutteranno i collegamenti esistenti fra le tabelle.
Invoice
Invoice_Number
Customer_Number
Payment
INTEGER

INTEGER
DECIMAL (10, 2)
11731
3175
0
11732
3176
249.50
11733
3175
0
Item
Invoice_Number
Product_Code
Quantity
INTEGER
CHAR (10)
INTEGER
11731
116-064
3
11731
257-535
1
Figura 6
11731
643-119
2
Tabelle collegate
11732
116-064
10
per fatture e articoli,
per realizzare
11733
116-064
2
una relazione “uno

11733
643-119
1
a molti”
Connessione a base di dati
9
Figura 7
Collegamenti tra
le tabelle della base
di dati d’esempio
Consigli per la produttività 3
Non replicate colonne in una tabella

Se vi accorgete che state numerando le colonne di una tabella con i suffissi 1, 2, e così via (come Quantity1, Quantity2, Quantity3 nell’esempio precedente), allora siete probabilmente sulla strada sbagliata. Come potete sapere che ci saranno esattamente tre quantità? In questo caso, è giunto il momento di creare una nuova tabella.

Aggiungete una tabella che contenga le informazioni che avevate replicato nelle colonne, inserendo un’ulteriore colonna alla tabella che la colleghi a una chiave presente nella prima tabella, come il numero della fattura nel nostro esempio. Usando una tabella aggiuntiva, potete realizzare una relazione “uno a molti”.

2
Interrogazioni

Ipotizziamo che le tabelle della nostra base di dati siano già state create e che vi siano stati inseriti dei dati. Una volta che una base di dati contiene delle informazioni, vorrete interrogare (query) la base di dati per ottenere le informazioni che contiene, come, ad esempio:

- *Quali sono i nomi e gli indirizzi di tutti i clienti?*
- *Quali sono i nomi e gli indirizzi di tutti i clienti della California?*
- *Quali sono i nomi e gli indirizzi di tutti i clienti che hanno acquistato un tostapane?*

10
Connessione a base di dati

Invoice
Item
Invoice_
Customer_
Payment
Invoice_
Product_
Quantity
Number
Number
Number
Code
INTEGER
INTEGER
DECIMAL (10, 2)
INTEGER
CHAR (10)
INTEGER
11731
3175
0
11731
116-064
3
11732
3176
249.50
11731
257-535
1
11733
3175
0
11731
643-119
2
11732
116-064

10
11733
116-064
2
11733
643-119
1
Customer
Customer_
Name
Address
City
State
Zip
Number
INTEGER
CHAR (40)
CHAR (40)
CHAR (30)
CHAR (2)
CHAR (10)
3175
Sam’s Small
100 Main Street
Anytown
CA
98765
Appliances
3176
Electronics
1175 Liberty Ave
Pleasantville
MI
45066
Unlimited
Product_Code
Description
Unit_Price

CHAR (10)
CHAR (20)
DECIMAL (10, 2)
116-064
Toaster
24.95
Figura 8
257-535
Hair dryer
29.95
Base di dati usata
643-119
Car vacuum
19.99
come esempio

■ *Quali sono i nomi e gli indirizzi di tutti i clienti che hanno fatture non pagate?*

■ *…*

In questo paragrafo apprenderete come si formulano semplici e complesse interrogazioni in SQL; per i nostri esempi, useremo i dati della Figura 8.

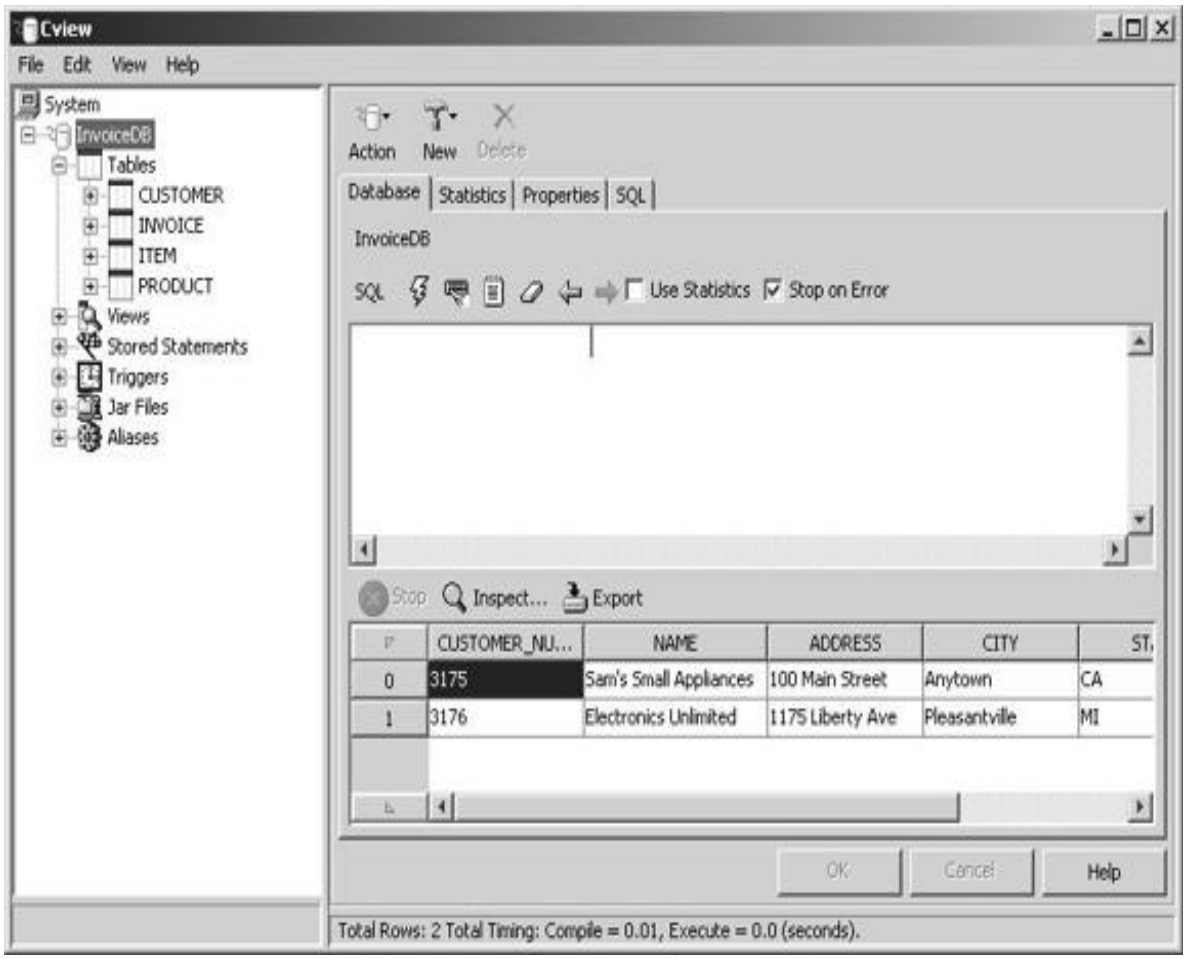
2.1

Interrogazioni semplici

Per interrogare una base di dati,

Per eseguire interrogazioni in SQL, si usa il comando SELECT. Ad esem-usate l'enunciato SELECT.

*pio, il comando per selezionare tutti i dati della tabella Customer è SELECT * FROM Customer*



Connessione a base di dati

11

Il risultato è

Customer_

Name

Address

City

State

Zip

Number

3175

Sam’s Small

100 Main

Anytown

CA

98765

Appliances

Street
3176
Electronics
1175 Liberty Pleasantville
MI
45066
Unlimited
Ave

Il risultato prodotto da un'interrogazione è una vista (view): un insieme di righe e colonne che fornisce una "finestra" attraverso la quale si possono osservare i dati della base di dati. Se selezionate tutte le righe e tutte le colonne di una tabella, ovviamente ottenete proprio una vista dell'intera tabella.

Molti sistemi di basi di dati forniscono strumenti che consentono l'invio interattivo di comandi SQL; un esempio tipico è mostrato in Figura 9. Quando inviate un comando SELECT, lo strumento visualizza la vista risultante. A questo punto potete andare avanti al Paragrafo 5 e installare una base di dati, oppure può darsi che il vostro computer in laboratorio abbia una base di dati già installata. Successivamente, potete eseguire lo strumento SQL interattivo della vostra base di dati e provare per conto vostro alcune interrogazioni.

2.2

Selezionare colonne

Spesso non siete interessati a tutte le colonne di una tabella: supponete che il vostro commesso viaggiatore stia pianificando un viaggio per contattare tutti i clienti. Per ***Figura 9***

Uno strumento SQL
interattivo

pianificare il percorso, vuole sapere soltanto la città e lo stato di tutti i clienti; ecco, quindi, l'interrogazione:

SELECT City, State FROM Customer

Il risultato è

City
State
Anytown

CA
Pleasantville
MI

Come potete vedere, la sintassi per selezionare colonne è intuitiva: specificate semplicemente i nomi delle colonne che volete, separati da virgole.

2.3

Selezionare sottoinsiemi

*Avete appena visto come potete restringere una vista in modo che mostri alcune colonne da voi selezionate. Altre volte vorrete selezionare alcune righe che soddisfino un particolare criterio: ad esempio, può darsi che vogliate cercare tutti i clienti resi-denti in California. Quando volete selezionare un sottoinsieme, usate la clausola WHERE, seguita dalla condizione che descrive il sottoinsieme. Ecco un esempio: SELECT * FROM Customer WHERE State = 'CA'*

Il risultato è

Customer_Number
Name
Address
City
State
Zip
3175
Sam's Small
100 Main Street
Anytown
CA
98765
Appliances

Dovete fare un po' di attenzione quando esprimete la condizione nella clausola WHERE, perché la sintassi SQL è diversa dalla sintassi Java. Come già sapete, in SQL si usano singoli apici per delimitare stringhe, come 'CA'. Inoltre, si usa un singolo segno

*=, e non un doppio segno ==, per verificare l'uguaglianza. Per verificare la diversità, usate l'operatore <>. Ad esempio, il comando SELECT * FROM Customer WHERE State <> 'CA'*

seleziona tutti i clienti che non si trovano in California.

Con l'operatore LIKE potete cercare la corrispondenza con determinati modelli (pattern). La parte a destra dell'operatore deve essere una stringa che contiene i simboli speciali _ (corrispondenza con uno e un solo carattere) o % (corrispondenza con qualsiasi sequenza di caratteri). Ad esempio, l'espressione Name LIKE '_o%'

corrisponde a tutte le stringhe il cui secondo carattere sia una "o". Quindi, "Toaster" è una corrispondenza, ma "Crowbar" non lo è.

Potete combinare espressioni con i connettori logici AND, OR e NOT (ma non usate gli operatori Java &&, || e !). Ad esempio

*SELECT **

FROM Product

WHERE Price < 100

AND Description <> 'Toaster'

seleziona tutti i prodotti che non siano tostapane e il cui prezzo sia inferiore a 100.

Ovviamente, potete selezionare contemporaneamente sottoinsiemi di righe e di colonne, come in questo esempio:

SELECT Name, City FROM Customer WHERE State = 'CA'

2.4

Calcoli

Supponete di voler trovare quanti clienti ci sono in California. Usate la funzione COUNT: SELECT COUNT(*) FROM Customer WHERE State = 'CA'

Oltre alla funzione COUNT, esistono altre quattro funzioni: SUM, AVG (calcola la media, average), MAX e MIN.

*Il simbolo * significa che volete effettuare il calcolo sulle intere righe della vista, cosa che è appropriata soltanto per la funzione COUNT. Per le altre funzioni, dovete accedere a una colonna specifica, inserendone il nome tra le parentesi: SELECT AVG(Price) FROM Product*

2.5

Unioni

Le interrogazioni che avete visto fino a ora hanno tutte coinvolto una sola tabella, ma solitamente le informazioni che volete sono distribuite su più tabelle. Ad esempio, supponete che vi sia chiesto di trovare tutte le fatture che contengono come articolo un aspirapolvere per automobile. Nella tabella Product, potete eseguire un'interrogazione per trovare il codice prodotto:

SELECT Product_Code
FROM Product
WHERE Description = 'Car vacuum'

Troverete che l'aspirapolvere per automobile ha il codice prodotto 643-119. Quindi, potete eseguire una seconda interrogazione:

SELECT Invoice_Number
FROM Item
WHERE Product_Code = '643-119'

ma avrebbe più senso combinare queste due interrogazioni in modo da non dover ***14***

Connessione a base di dati

tenere traccia del risultato intermedio. Combinando le interrogazioni, osservate che le due tabelle sono collegate dal campo Product_Code: vogliamo ottenere una vista delle righe corrispondenti in entrambe le tabelle. In altre parole, vogliamo restringere la ricerca alle righe dove

Product.Product_Code = Item.Product_Code

In questo caso, la sintassi che abbiamo utilizzato: NomeTabella. NomeColonna

indica una particolare colonna nella tabella designata. Quando un'interrogazione coinvolge più tabelle, bisogna specificare sia il nome della tabella, sia il nome della colonna, per cui l'interrogazione combinata è

SELECT Item.Invoice_Number
FROM Product, Item
WHERE Product.Description = 'Car vacuum'
AND Product.Product_Code = Item.Product_Code

Il risultato è

Invoice_Number
11731

11732

In questa interrogazione, la clausola FROM contiene i nomi di più tabelle, separati da virgole (l'ordine in cui vengono elencati non è importante): una tale interrogazione viene detta unione (join), perché prevede l'unione di più tabelle.

Una unione è un'interrogazione

Può succedere che vogliate sapere in quali città sono molto richiesti che coinvolge più tabelle.

gli asciugacapelli: per ottenere tale informazione, dovete aggiungere la tabella Customer all'interrogazione, poiché contiene gli indirizzi dei clienti. I clienti sono accessibili mediante le fatture, per cui avete bisogno anche di quella tabella. Ecco, quindi, l'interrogazione completa: SELECT Customer.City, Customer.State, Customer.Zip FROM Product, Item, Invoice, Customer

WHERE Product.Description = 'Hair dryer'

AND Product.Product_Code = Item.Product_Code

AND Item.Invoice_Number = Invoice.Invoice_Number AND Invoice.Customer_Number = Customer.Customer_Number Il risultato è

City

State

Zip

Anytown

CA

98765

Connessione a base di dati

15

Ogni volta che formulate un'interrogazione che coinvolge più tabelle, ricordate di fare tutte le seguenti cose:

■ *Elencate nella clausola FROM tutte le tabelle che sono coinvolte nell'interrogazione.*

■ *Usate la sintassi NomeTabella. NomeColonna per fare riferimento ai nomi delle colonne.*

■ *Elencate nella clausola W H E R E tutte le condizioni dell'unione (NomeTabella1. NomeColonna1 = NomeTabella2. NomeColonna2).*

Come potete vedere, queste interrogazioni sono un po' complesse, ma le basi di dati sono assolutamente in grado di rispondere (consultate i Consigli per la Produttività 4). Un'interessante caratteristica del linguaggio SQL è che dice alla base di dati che cosa volete, non come deve fare per trovare la risposta: è compito della base di dati identificare una piano d'azione per trovare la risposta alla vostra interrogazione nel minor numero di passi. I produttori di basi di dati commerciali sono molto orgogliosi quando trovano modi efficienti per velocizzare le interrogazioni: strategie di ottimiz-zazione delle interrogazioni, conservazione dei risultati di interrogazioni precedenti, e così via. A questo proposito, il linguaggio SQL è molto diverso dal linguaggio Java: gli enunciati SQL sono descrittivi e lasciano alla base di dati la scelta di come eseguirli, mentre gli enunciati Java sono prescrittivi, nel senso che indicano chiaramente i passi che il vostro programma deve eseguire.

Errori comuni 1

Unione di tabelle senza la specifica di una condizione

Se selezionate dati da più tabelle senza eseguire una restrizione, il risultato è in qualche modo sorprendente: ottenete un insieme contenente tutte le combinazioni dei valori, indipendentemente dal fatto che alcune combinazioni esistano realmente nell'insieme dei dati. Ad esempio, l'interrogazione SELECT Invoice.Invoice_Number, Customer.Name

FROM Invoice, Customer

produce l'insieme risultante

Invoice.Invoice_Number

Customer.Name

11731

Sam's Small Appliances

11732

Sam's Small Appliances

11733

Sam's Small Appliances

11731

Electronics Unlimited

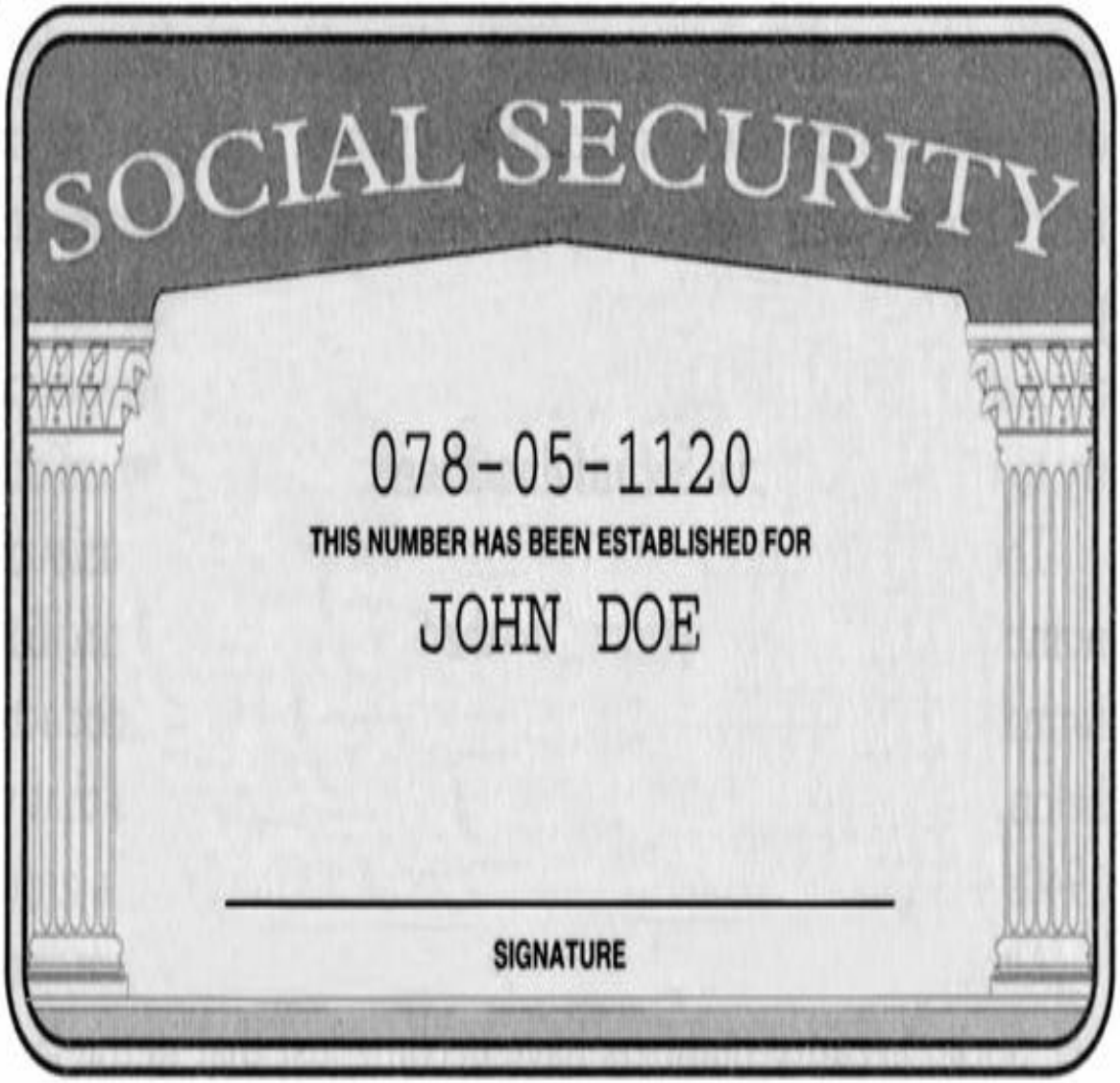
11732
Electronics Unlimited
11733
Electronics Unlimited
16
Connessione a base di dati
Come potete verificare, l'insieme risultante contiene tutte le sei combinazioni di numeri di fattura (11731, 11732, 11733) e di nomi di clienti (Sam's Small Appliances e Electronics Unlimited), anche se tre di tali combinazioni non ricorrono nelle reali fatture. Per ridurre l'insieme delle combinazioni, è necessario fornire una clausola WHERE; ad esempio
SELECT Invoice.Invoice_Number, Customer.Name
FROM Invoice, Customer
WHERE Invoice.Customer_Number = Customer.Customer_Number produce
Invoice.Invoice_Number
Customer.Name
11731
Sam's Small Appliances
11732
Electronics Unlimited
11733
Sam's Small Appliances
2.6
Aggiornare ed eliminare dati
I comandi UPDATE e DELETE
Fino a ora avete visto come formulare interrogazioni SELECT sempre modificano i dati nella base di dati.
più complesse: il prodotto di un'interrogazione SELECT è un insieme risultante che potete osservare e analizzare. Esistono due tipi di enunciati correlati, UPDATE e DELETE, che non producono un insieme risultante, ma modificano la base di dati. L'enunciato DELETE è il più facile dei due: semplicemente, elimina le righe che specificate. Ad esempio, per eliminare tutti i clienti della California, inviate l'enunciato seguente:
DELETE FROM Customer WHERE State = 'CA'

L’interrogazione UPDATE vi consente di aggiornare le colonne di tutte le righe che sod-disfano una determinata condizione. Ad esempio, ecco come potete aggiungere un’altra unità alla quantità di ciascun articolo della fattura numero 11731.

UPDATE Item
SET Quantity = Quantity + 1
WHERE Invoice_Number = ‘11731’

Potete aggiornare i valori di più colonne specificando più espressioni di aggiornamento nella clausola SET, separate da virgole.

Entrambi gli enunciati, DELETE e UPDATE, restituiscono un valore: il numero di righe che vengono eliminate o aggiornate.



Note di cronaca 1

Basi di dati e privacy

La maggior parte delle aziende usa i computer per conservare enormi archivi di dati contenenti annotazioni sui clienti e altre informazioni di lavoro. Le basi di dati non solo servono a ridurre il costo del lavoro, ma anche a migliorare la qualità del servizio offerto dalle aziende. Oggi è quasi inimmaginabile quanto tempo ci volesse in passato per prelevare denaro da una banca o per prenotare un viaggio.

Molte aziende raccolgono grandi quantità di dati che riguardano i loro clienti e spesso li condividono con altre aziende. Tali dati non vengono usati soltanto per fornire un buon servizio ai clienti, ma anche per altri scopi commerciali: ad esempio, per indirizzare meglio la pubblicità. Una ditta potrebbe avere un archivio con gli indirizzi dei proprietari di automobili e un archivio di persone con buoni precedenti in quanto ai pagamenti; potrebbe, quindi, inviare offerte speciali a tutti i suoi clienti che hanno inviato un ordine nell'ultimo mese, hanno un'automobile costosa e pagano regolar-mente le proprie fatture.

Questo genere di interrogazioni, naturalmente, è molto più veloce se tutti gli archivi che riguardano i clienti usano la stessa chiave, e questo è il motivo per cui tante organizzazioni negli Stati Uniti cercano di ottenere i numeri della previdenza sociale dei loro clienti.

Il Social Security Act del 1935 prevede che a ogni contribuente venga assegnato un numero per la registrazione dei contributi al fondo della previdenza sociale. Questi numeri hanno un formato caratteristico, come 078-05-1120. (Questo particolare numero non è un numero della previdenza sociale appartenente di fatto a una persona.

È stato stampato sulle tessere di esempio inserite nei portafogli negli anni '40 e '50.) La Figura 10 mostra una tessera della previdenza sociale.

Nonostante in origine non dovessero essere utilizzati come numeri di identificazione universale, alla fine degli anni '60 i

numeri della previdenza sociale sono diven-tati esattamente questo. Il fisco e molti altri istituti pubblici sono autorizzati a racco-gliere tali numeri, così come le banche (per la notifica degli interessi maturati) e, naturalmente, i datori di lavoro. Anche molte altre organizzazioni trovano comodo usare quel numero.

Figura 10

Una tessera della previdenza sociale statunitense

18

Connessione a base di dati

Dal punto di vista tecnico, i numeri della previdenza sociale rappresentano un metodo scadente per indicizzare una base di dati. C'è il rischio di avere due righe con lo stesso numero, perché molti immigranti illegali usano numeri falsi. Non tutti pos-siedono un numero, in particolare i clienti stranieri non lo hanno. Dato che non esiste una cifra di controllo (checksum), un errore di trascrizione (come lo scambio di due cifre) non è individuabile. (I numeri delle carte di credito hanno, invece, una cifra di controllo.) Per la stessa ragione, è facile per chiunque inventarsi un numero.

Alcune persone si preoccupano molto per il fatto che quasi tutte le organizzazioni vogliono registrare il loro numero della previdenza sociale. A meno che non ci sia un'esigenza lecita, come per le banche, di solito ci si può opporre o rivolgersi ad altri.

Anche quando un'organizzazione è autorizzata a conoscere il numero, come il datore di lavoro, si può pretendere che il numero venga utilizzato solo a scopi fiscali e sui documenti della previdenza sociale e non sul fronte di una tessera di identificazione.

Purtroppo, in genere ci vuole uno sforzo quasi sovrumano per trovare in un'organizzazione qualcuno che abbia l'autorità di accettare documenti privi di un numero di previdenza sociale o di assegnare un diverso numero di identificazione.

A mano a mano che Internet diventa sempre più parte integrante delle nostre vite, potete assistere in prima persona al

vorace appetito di informazioni personali da parte di addetti alla pubblicità di prodotti e servizi. Avrete molte opportunità di ottenere servizi gratuiti (come la posta elettronica), se siete disposti a divulgare alcune informazioni personali e se vi prestate al monitoraggio delle vostre abitudini di ricerca e di acquisti on line.

Il disagio che provano molte persone nei confronti della informatizzazione delle loro informazioni personali è comprensibile: esiste la possibilità che le aziende e il governo uniscano varie basi di dati e ne ricavino informazioni sul nostro conto che potremmo preferire che non avessero o che potrebbero semplicemente non essere vere. Una compagnia di assicurazioni potrebbe negarvi la copertura o pretendere un premio più alto, se scoprisse che avete troppi parenti con una certa malattia. Potrebbe venirvi negato un incarico a causa di false informazioni su di voi e sulla vostra salute, senza che voi neppure possiate conoscerne la ragione. Questi sono sviluppi molto preoccupanti, che hanno avuto un impatto molto negativo su un numero piccolo ma crescente di persone. Leggete [3] e [4] per saperne di più.

3

Installare una base di dati

Sono disponibili molti sistemi per basi di dati, fra i quali:

- *Basi di dati commerciali a elevate prestazioni, come Oracle, DB2, Informix, Sybase e Microsoft SQL Server*
- *Basi di dati “open-source” (con il codice sorgente disponibile), come PostgreSQL, MySQL e Interbase*
- *Basi di dati “snelle” (lightweight) in Java, come Cloudscape e Pointbase*
- *Basi di dati per computer da scrivania, come Paradox e Access Quale si dovrebbe scegliere per imparare la programmazione di basi di dati? Questo dipende fortemente dalla vostra disponibilità di denaro, di risorse di calcolo e di espe-Connessione a base di dati*

rienza nell’installazione di software complesso. In un laboratorio con un gestore ben preparato, ha molto senso installare una base di dati commerciale come Oracle, ma tali

prodotti e l'hardware necessario per eseguirli, nonché il personale per gestirli, sono costosi. Le alternative "open-source" sono disponibili gratuitamente e la loro qualità è molto migliorata negli ultimi anni: sebbene non siano adatte per applicazioni di grandi dimensioni, funzionano bene a scopo didattico, anche se la loro installazione non è adatta ai deboli di cuore. Le basi di dati "snelle" in Java sono molto più facili da installare e funzionano su molte piattaforme diverse: sono molto adatte ai principianti, ma hanno alcuni svantaggi, perché solitamente garantiscono prestazioni inferiori e spesso non permettono l'esecuzione di comandi SQL complessi. Le basi di dati per computer da scrivania, come Access, sono generalmente una scelta di scarsa qualità e si dovrebbero evitare, se possibile.

Se lavorate in un laboratorio di informatica, qualcuno installerà per voi una base di dati e dovreste essere in grado di trovare le istruzioni per utilizzarla. Se installate una base di dati per conto vostro, vi consigliamo di iniziare con una base di dati "snella" in Java, come Cloudscape, che mette a disposizione gratuitamente una versione di prova per sessanta giorni (http://www.cloudscape.com).

Le istruzioni dettagliate per installare una base di dati variano molto: ecco, però, una sequenza generica dei passi necessari per installare una base di dati e per collaudare la vostra installazione.

1. Prendete possesso di un programma per basi di dati, su un CD-ROM o scarican-dolo dal Web.

2. Leggete le istruzioni di installazione.

3. Installate il programma, in un modo che può essere molto diverso da un programma a un altro. Può darsi che sia semplice come l'esecuzione di un programma di installazione, oppure che dobbiate compilare il programma a partire dal suo codice sorgente.

4. Fate partire la base di dati. Nella maggior parte dei sistemi di basi di dati (ma non con alcuni dei sistemi snelli di basi di dati in Java), dovete far partire il server della base di dati prima di eseguire qualsiasi operazione su di essa. Leggete le istruzioni di installazione per i dettagli.

5. Approntate i ruoli degli utenti. Questo passo prevede tipicamente l'esecuzione di un programma di gestione, la connessione a esso come gestore con un ruolo predefinito e l'aggiunta di nomi di utenti e delle relative parole di accesso. Se siete l'unico utente della base di dati, può darsi che possiate usare il solo ruolo predefinito. Di nuovo, i dettagli sono molto variabili da una base di dati a un'altra, e dovete consultare la documentazione.

6. Eseguite una prova di collaudo. La maggior parte delle basi di dati viene fornita insieme a un programma che vi consente di eseguire istruzioni SQL interattiva-mente: identificate tale programma e scoprite come collegarvi alla base di dati nel ruolo di utente, quindi eseguite le seguenti istruzioni SQL: CREATE TABLE Test (Name CHAR(20))

INSERT INTO Test VALUES ('Romeo')

*SELECT * FROM Test*

DROP TABLE Test

20

Connessione a base di dati

A questo punto dovreste ottenere una visualizzazione della base di dati Test che mostra una sola riga e una sola colonna, contenente la stringa "Romeo". In caso contrario, leggete attentamente la documentazione del vostro strumento SQL per vedere come inserire enunciati SQL. Ad esempio, in alcuni strumenti di questo tipo c'è bisogno di uno speciale terminatore per ciascun enunciato SQL.

Per accedere a una base di dati

Una volta che la base di dati è allestita e funziona correttamente, dovete da un programma Java avete bi-installare un driver JDBC. L'acronimo JDBC significa Java Database Con-sogno di un driver JDBC (Java

nectivity, che è il nome di una tecnologia che consente a programmi Database Connectivity).

Java di interagire con basi di dati. Diverse basi di dati richiedono driver diversi, che possono essere forniti dal produttore della base di dati o da un produttore diverso: in ogni caso, dovete trovare e installare il driver appropriato per la vostra base di dati. Quando il vostro programma Java invia

comandi SQL, il driver li rimanda alla base di dati e consente al vostro programma di analizzare i risultati (osservate la Figura 11).

Il Java Software Development Kit contiene un driver denominato JDBC/ODBC, che può essere utilizzato per connettersi a qualsiasi base di dati che rispetti lo standard ODBC (Open Database Connectivity); quando la tecnologia JDBC è stata presentata, tale driver di collegamento fu molto utile, perché in quel momento la maggior parte delle basi di dati aveva un driver ODBC, ma oggi quasi tutte le basi di dati hanno un driver JDBC più potente e più facile da configurare del driver JDBC/ODBC. Per questo motivo, vi consigliamo di usare un driver JDBC invece di un driver ODBC

insieme al driver JDBC/ODBC.

Accertatevi che il driver JDBC si

Dopo aver installato il driver JDBC, dovreste eseguire un program-trovi nel percorso delle classi

ma di prova per verificare che l'installazione sia andata a buon fine. Per (classpath) prima di eseguire il

questo programma di prova, ipotizziamo che abbiate già creato una ta-programma Java.

bella Test nella vostra base di dati, come descritto nelle istruzioni di prova della base di dati stessa. Ecco i passi necessari per collaudare il driver JDBC.

1. Ogni driver JDBC contiene una parte di codice Java che è necessario ai vostri programmi Java per connettersi alla base di dati. Consultando la documentazione del driver JDBC, identificate quale sia il percorso delle classi del driver. Ecco un esempio tipico, che riguarda il percorso delle classi per il driver JDBC di Cloudscape, installato sul disco C di un computer dotato del sistema operativo Windows.

C:\cloudscape\lib\cloudscape.jar

Una particolare versione della base di dati Oracle usa, ad esempio, il seguente percorso:

/usr/local/oracle/jdbc/classes111b.zip

Figura 11

Architettura JDBC

Connessione a base di dati

Troverete questa informazione nella documentazione del driver JDBC.

2. Identificate il nome della classe del driver che dovete caricare. Ecco una tipica classe del driver, sempre relativa alla base di dati Cloudscape: COM.cloudscape.core.JDBCDriver

La base di dati Oracle usa, invece, il driver

oracle.jdbc.driver.OracleDriver

In generale, vi dovete aspettare che il nome della classe si trovi in un pacchetto il cui nome corrisponde a quello del produttore della base di dati (come COM.cloudscape oppure oracle), seguito da un nome di classe specifico del produttore. La documentazione del vostro driver JDBC conterrà questa informazione.

3. Identificate il nome dello URL della base di dati che il vostro driver si aspetta di ricevere. Tutti gli URL delle basi di dati hanno questo formato: jdbc:sottoprotocollo:dati specifici del driver

Il sottoprotocollo è un codice che identifica il produttore del driver, come cloudscape o oracle. I dati specifici del driver codificano il nome e la collocazione della base di dati. Ecco due esempi tipici:

jdbc:cloudscape:InvoiceDB;create=true

jdbc:oracle:thin:@larry.mathcs.sjsu.edu:1521:InvoiceDB

Di nuovo, consultate la documentazione del vostro driver JDBC per i dettagli riguardanti il formato dello URL della base di dati e le modalità previste per specificare la base di dati che usate.

4. Nel programma TestDB.java che trovate alla fine di questo paragrafo, modificate il file database.properties, fornendo

- *Il nome della classe del driver*
- *Lo URL della base di dati*
- *Il vostro nome utente nella base di dati*
- *La vostra parola d'accesso alla base di dati*

5. Compilate il programma in questo modo:

javac TestDB.java

6. Eseguite il programma in questo modo:

java -classpath percorso_della_classe_del_driver;. TestDB

database.properties

Nel sistema operativo Unix/Linux, usate invece il separatore : nel percorso delle classi:

java -classpath percorso_della_classe_del_driver:. TestDB database.properties

Se tutto funziona correttamente, otterrete in uscita un elenco di tutti i dati contenuti nella tabella Test: se avete seguito alla lettera le istruzioni, vedrete una riga in uscita contenente il nome “Romeo”.

Se il vostro programma non funziona, le cause possono essere diverse.

• Se ricevete un messaggio d’errore relativo a un driver mancante, ciò significa che il programma non è in grado di caricare il driver JDBC. Controllate il percorso delle classi e il nome del driver.

• Se ricevete un messaggio d’errore del driver che non riesce a connettersi alla base di dati, allora la base di dati non è in esecuzione, oppure, se si trova su un diverso computer, non è raggiungibile. Verificate che la base di dati sia funzio-nante connettendovi con uno strumento SQL.

• Se ricevete un messaggio d’errore relativo al fallimento dell’autenticazione dell’utente, verificate il nome della base di dati, il nome dell’utente e la parola d’accesso contenute nello URL della base di dati.

• Se ricevete un messaggio d’errore relativo alla mancanza della tabella Test, ac-certatevi di aver creato la tabella e di avervi inserito dei dati, come descritto nelle istruzioni di collaudo appena fornite.

Ecco il programma di prova, le cui istruzioni Java verranno illustrate nel paragrafo successivo.

File TestDB.java

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
/**
```

Collauda l'installazione di una base di dati creando una semplice tabella e interrogandola. Invocate il programma in questo modo: java -classpath percorso_della_classe_del_driver;. TestDB

```
database.properties
*/
public class TestDB
{
public static void main(String[] args)
{
if (args.length == 0)
{
System.out.println("Usage: TestDB propertiesFile");
System.exit(0);
}
else
SimpleDataSource.init(args[0]);
Connection conn = SimpleDataSource.getConnection();
Statement stat = conn.createStatement();
stat.execute("CREATE TABLE Test (Name CHAR(20))");
stat.execute("INSERT INTO Test VALUES ('Romeo')");
```

Connessione a base di dati


```
ResultSet result = stat.executeQuery(
"SELECT * FROM Test");
result.next();
System.out.println(result.getString("Name"));
result.close();
stat.execute("DROP TABLE Test");
stat.close();
conn.close();
}
}
```

File SimpleDataSource.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.io.FileInputStream;
```

```
import java.io.IOException;
import java.util.Properties;
/**
Una semplice fonte di dati per stabilire connessioni a basi di dati.
*/
public class SimpleDataSource
{
/**
Inizializza la fonte di dati.
@param fileName il nome del file contenente le proprietà relative al driver, allo URL, al nome utente
e alla parola di accesso alla base di dati
*/
public static void init(String fileName)
throws IOException, ClassNotFoundException
{
Properties props = new Properties();
FileInputStream in = new FileInputStream(fileName); props.load(in);
String driver = props.getProperty("jdbc.driver"); url = props.getProperty("jdbc.url");
username = props.getProperty("jdbc.username");
password = props.getProperty("jdbc.password");
Class.forName(driver);
}
/**
Stabilisce una connessione alla base di dati.
@return la connessione alla base di dati
*/
```



```
Connessione a base di dati
public static Connection getConnection()
throws SQLException
{
return DriverManager.getConnection(url,
username, password);
}
```

private static String url;
private static String username;
private static String password;
}

File database.properties di esempio

jdbc.driver=COM.cloudscape.core.JDBCDriver
jdbc.url=jdbc:cloudscape:BankDB:create=true
jdbc.username=username
jdbc.password=password

4
Programmazione di basi di dati in Java

4.1
Connessione alla base di dati

Per accedere a una base di dati

Per connettervi a una base di dati, avete bisogno di un esemplare della da un programma Java usate un

classe Connection: il frammento di codice che segue vi mostra come oggetto di tipo Connection.

ottenerlo. Dapprima caricate il driver della base di dati, quindi richiedete una connessione al DriverManager, fornendo alle stringhe driver, url, username e password i valori che si riferiscono alla vostra base di dati.

String driver = ...;
String url = ...;
String username = ...;
String password = ...;
Class.forName(driver); // carica il driver
Connection conn = DriverManager.getConnection(url, username, password);

Quando avete terminato di inviare comandi alla base di dati, dovreste chiudere la connessione invocando il metodo close:

conn.close();

Si tratta di una visione alquanto semplificata della gestione di una connessione: in realtà, possono accadere due problemi. I programmi più grandi, come l'esempio della banca del Paragrafo 5, hanno bisogno di effettuare connessioni alla base di dati da molte classi, e non volete propagare a tutte queste

classi le informazioni necessarie per collegarsi alla base di dati. Ancora, solitamente non è praticabile l'utilizzo di un'unica Connessione a base di dati

25

connessione per tutte le richieste alla base di dati: ad esempio, il contenitore usato dalla tecnologia Java Server Pages (JSP) può gestire molte richieste simultanee di una stessa pagina Web provenienti da diversi browser e ciascuna richiesta di pagine ha bisogno della propria connessione alla base di dati. Poiché, però, aprire una connessione alla base di dati è un'operazione abbastanza lenta e le richieste di pagine arrivano molto frequentemente, le connessioni devono essere conservate e riutilizzate, anziché chiuse e riaperte. I dettagli di queste operazioni possono essere complessi e, attualmente, non esiste un'implementazione standard disponibile.

Disaccoppiare la gestione delle connessioni dal rimanente codice della base di dati è sempre una buona idea: per questo motivo, usiamo la classe SimpleDataSource, di cui trovate l'implementazione alla fine del paragrafo precedente e che rappresenta uno strumento molto semplice per la gestione della connessione. All'inizio del programma, ne invocate il metodo statico init con il nome del file di configurazione della base di dati, come in questo esempio:

SimpleDataSource.init("database.properties");

Il file di configurazione è un file in formato testo che contiene quattro righe jdbc.driver=...

jdbc.url=...

jdbc.username=...

jdbc.password=...

Il metodo init usa la classe Properties, che è stata progettata per rendere agevole la lettura di un file di questo tipo.

La classe Properties ha un metodo load per leggere un file di coppie chiave/valore da un flusso:

Properties props = new Properties();

FileInputStream in = new FileInputStream(fileName); props.load(in);

Il metodo getProperty restituisce il valore di una chiave: String driver = props.getProperty("jdbc.driver"); In realtà, non

dovete pensarci: di questi dettagli si occupa il metodo init.

Ogni volta che avete bisogno di una connessione, invocate Connection conn = SimpleDataSource.getConnection(); Quando avete terminato, dovete chiudere la connessione invocando conn.close();

I veri gestori di connessioni hanno metodi un po' diversi, ma il principio di base è lo stesso.

26

Connessione a base di dati

Suggerimenti per la qualità 1

Non scrivete nel programma i parametri di connessione

Scrivere nel programma i parametri di connessione alla base di dati è considerato poco elegante.

```
public class MyProg
{
public static void main(String[] args)
{
// non fatelo
String driver = "COM.cloudscape.core.JDBCDriver"; String url = "jdbc:cloudscape:InvoiceDB;create=true"; String username = "username";
String password = "password";
...
}
}
```

Se volete, poi, passare a una diversa base di dati, dovete trovare queste stringhe, ag-giornarle e compilare nuovamente il programma.

Al contrario, inserite le stringhe in un file di configurazione. Il file SimpleDataSource.java legge un file di configurazione contenente i parametri della connessione alla base di dati: per connettervi a una base di dati diversa, fornite semplicemente sulla riga dei comandi un diverso nome di file di configurazione.

4.2 Esecuzione di enunciati SQL

Un oggetto di tipo Connection

Una volta che avete una connessione, la potete usare per creare og-può creare oggetti di tipo Sta-

getti di tipo Statement, di cui avete bisogno per eseguire enunciati tement, che si usano per ese-SQL.

guire comandi SQL.

Statement stat = conn.createStatement();

Il metodo execute della classe Statement esegue un enunciato SQL. Ad esempio stat.execute("CREATE TABLE Test (Name CHAR(20))"); stat.execute("INSERT INTO Test VALUES ('Romeo')"); Il risultato di un'interrogazione

Per inviare un'interrogazione, usate il metodo executeQuery della clas-SQL viene restituito in un ogget-

se Statement; il risultato dell'interrogazione viene restituito come og-to di tipo ResultSet.

getto di tipo ResultSet. Ad esempio

*String queryStatement = "SELECT * FROM Test";*

ResultSet result = stat.executeQuery(queryStatement); Connessione a base di dati

Nel prossimo paragrafo vedrete come usare l'oggetto di tipo ResultSet per analizzare il risultato dell'interrogazione.

Per gli enunciati UPDATE, potete usare il metodo executeUpdate, che restituisce il numero di righe interessate dall'aggiornamento:

String updateStatement = "UPDATE Item "

+ "SET Quantity = Quantity + 1 "

+ "WHERE Invoice_Number = '11731'";

int count = stat.executeUpdate(updateStatement); In alternativa, potete usare il generico comando execute anche per eseguire interrogazioni e aggiornamenti: esso restituisce un valore di tipo boolean per indicare se il comando SQL fornisce un insieme di dati come risultato. In tal caso, potete recuperar-lo con il metodo getResultSet, oppure potete conoscere il conteggio degli aggiornamenti effettuati usando il metodo getUpdateCount.

String command = ...;

boolean hasResultSet = stat.execute(command);

if (hasResultSet)

```
{
ResultSet result = stat.getResultSet();
...
}
else
{
int count = stat.getUpdateCount();
...
}
```

Potete riutilizzare un oggetto di tipo Statement per eseguire tutti i comandi SQL che volete, ma, per ciascun oggetto di tipo Statement, dovete avere attivo un solo oggetto di tipo ResultSet per volta. Se il vostro programma ha bisogno di consultare diversi insiemi di risultati contemporaneamente, allora dovete creare diversi oggetti di tipo Statement.

Quando avete terminato di consultare un oggetto di tipo ResultSet, dovreste chiuderlo prima di inviare una nuova interrogazione allo stesso oggetto di tipo Statement.

```
result.close();
```

Quando avete finito di utilizzare un oggetto di tipo Statement, dovreste chiuderlo, chiudendo così automaticamente anche l'insieme di risultati a esso associato.

```
stat.close();
```

4.3

Analisi dei risultati di interrogazioni

Un oggetto di tipo ResultSet vi consente di recuperare i risultati, una riga alla volta.

Iterate lungo le righe, e per ogni riga potete esaminare i valori delle colonne; come gli iteratori delle liste, anche la classe ResultSet ha un metodo next per accedere alla riga successiva, ma il suo comportamento è un po' diverso. Il metodo next non restituisce

dati, ma un valore di tipo boolean che indica se sono disponibili ulteriori dati. Inoltre, quando ricevete un insieme di risultati dal metodo executeQuery, non vi è alcun dato disponibile: dovete invocare il metodo next per accedere alla

prima riga di dati. Sembra strano, ma ciò rende più semplice il ciclo di iterazione: while (result.next())

{

esamina le colonne dei dati della riga attuale

}

Se l'insieme di risultati è vuoto, la prima invocazione di result.next() restituisce false e il corpo del ciclo non viene mai eseguito. Altrimenti, la prima invocazione di result.next() recupera dalla base di dati i dati della prima riga. Come potete vedere, il ciclo termina quando il metodo next restituisce false, cioè quando tutte le righe sono state recuperate.

Dopo che l'oggetto che rappresenta l'insieme dei risultati ha recuperato una particolare riga, ne potete esaminare le colonne; diversi metodi di tipo get forniscono i valori delle colonne con il formato di numero, di stringa, di data, e così via. Uno di tali metodi ha un parametro di tipo intero che indica la posizione della colonna; un altro ha un parametro di tipo stringa che indica il nome della colonna. Ad esempio, potete esaminare il codice prodotto in questo modo

String productCode = result.getString(1);

oppure

String productCode = result.getString("Product_Code"); Notate che l'indice di tipo intero parte da uno, non da zero: getString(1) esamina la prima colonna. Gli indici delle colonne delle basi di dati sono diversi dagli indici degli array.

Accedere a una colonna usando un indice intero è leggermente più veloce ed è decisamente accettabile se avete indicato esplicitamente il nome delle colonne desi-derate nell'enunciato SELECT, come in questo caso: SELECT Invoice_Number FROM Invoice WHERE Payment = 0

*Se, invece, avete eseguito un'interrogazione di tipo SELECT *, è meglio usare un nome di colonna invece di un indice numerico: il vostro codice sarà più facile da leggere e non lo dovrete aggiornare quando modificherete la disposizione delle colonne.*

Nell'esempio precedente avete visto in azione il metodo getString. Per recuperare, invece, un numero, usate i metodi

getInt e getDouble, come in questi esempi: int quantity = result.getInt("Quantity");

double unitPrice = result.getDouble("Price");

Connessione a base di dati

29

Consigli per la produttività 4

Fate fare il lavoro alla base di dati

Adesso sapete come inviare interrogazioni SQL da un programma Java e come iterare nell'insieme risultante. Un errore compiuto comunemente dagli studenti è quello di scandire una tabella alla volta per trovare un risultato. Ad esempio, supponete di voler trovare tutte le fatture che contengono aspirapolvere per automobile. Potreste fare in questo modo:

*1. Inviate l'interrogazione SELECT * FROM Product e iterate nell'insieme risultante per trovare il codice prodotto dell'aspirapolvere per automobile 2. Inviate l'interrogazione SELECT * FROM Item e iterate nell'insieme risultante per trovare gli articoli aventi quel codice prodotto Tuttavia, questa procedura è estremamente inefficiente. Un tale programma esegue al rallentatore ciò che una base di dati esegue molto rapidamente, perché è stata progettata appositamente per farlo.*

Al contrario, dovreste lasciare che la base di dati faccia tutto il lavoro, inviando l'interrogazione completa:

SELECT Item.Invoice_Number

FROM Product, Item

WHERE Product.Description = 'Car vacuum'

AND Product.Product_Code = Item.Product_Code

Successivamente, iterate nell'insieme risultante per leggere i numeri delle fatture.

I principianti hanno spesso timore di inviare interrogazioni SQL complesse, ma se non sfruttate il linguaggio SQL rinunciate a uno dei maggiori benefici di una base di dati relazionali.

4.4

I dati descrittivi

di un insieme di risultati

I dati descrittivi (meta data) si ri-

Quando avete a disposizione un insieme di risultati che proviene da feriscono a un oggetto. I dati

una tabella sconosciuta, può darsi che vogliate sapere i nomi delle co-descrittivi di un insieme di risul-

lonne; per conoscere le proprietà di un insieme di risultati, potete usare tati descrivono le proprietà della classe ResultSetMetaData. Iniziate chiedendo all'insieme di risultati l'insieme stesso.

il suo oggetto con i dati descrittivi:

ResultSetMetaData metaData = result.getMetaData(); Quindi, potete conoscere il numero di colonne invocando il metodo getColumnCount; il metodo getColumnLabel fornisce il nome di ciascuna colonna; infine, il metodo getColumnDisplaySize restituisce l'ampiezza di una colonna, che è utile se volete stampare le righe della tabella con i dati incolonnati. Notate che gli indici di questi metodi par-tono da uno.

Ad esempio

```
for (int i = 1; i <= metaData.getColumnCount(); i++)
{
String columnName = metaData.getColumnLabel(i);
int columnSize = metaData.getColumnSize(i);
...
}
```

Alla fine di questo paragrafo troverete un utile programma che mette in pratica questi concetti. Il programma legge un file contenente enunciati SQL e li esegue tutti. Quando un enunciato produce un insieme di risultati, esso viene stampato, usando i suoi dati descrittivi per determinare il numero di colonne e i loro nomi.

Ad esempio, supponete di avere questo file:

File product.sql

```
CREATE TABLE Product (Product_Code CHAR(10),
Description CHAR(40), Price DECIMAL(10, 2))
INSERT INTO Product VALUES ('116-064', 'Toaster', 24.95)
INSERT INTO Product VALUES ('257-535', 'Hair dryer', 29.95)
```

INSERT INTO Product VALUES ('643-119', 'Car vacuum', 19.99)
*SELECT * FROM Product*

Quindi, eseguite il programma in questo modo:

java ExecSQL database.properties product.sql

Il programma esegue gli enunciati contenuti nel file e visualizza il risultato dell'interrogazione SELECT.

Potete anche usare il programma come strumento interattivo di collaudo. Eseguite in questo modo:

java ExecSQL database.properties

quindi digitate comandi SQL sulla riga di comando. Ogni volta che digitate il tasto Invio (Enter), il comando viene eseguito.

File ExecSQL.java

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.sql.SQLException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.Reader;
```



```
/**
Esegue tutti gli enunciati SQL presenti in un file.
Eseguite il programma in questo modo:
java -classpath percorso_della_classe_del_driver;. ExecSQL
database.properties commands.sql
*/
class ExecSQL
{
public static void main(String[] args)
throws Exception
{
if (args.length == 0)
{
```

```
System.out.println(
"Usage: java ExecSQL propertiesFile"
+ " [statementFile]");
System.exit(0);
}
SimpleDataSource.init(args[0]);
Connection conn = SimpleDataSource.getConnection();
Statement stat = conn.createStatement();
Reader reader;
if (args.length > 1)
reader = new FileReader(args[1]);
else
reader = new InputStreamReader(System.in);
BufferedReader in = new BufferedReader(reader);
boolean done = false;
while (!done)
{
String line = in.readLine();
if (line == null)
done = true;
else
{
try
{
boolean hasResultSet = stat.execute(line);
if (hasResultSet)
showResultSet(stat);
}
catch (SQLException exception)
{
exception.printStackTrace();
}
}
}
in.close();
stat.close();
```

```
conn.close();
}
/**
Visualizza un insieme di risultati.
@param stat l'enunciato di cui si vuole visualizzare l'insieme risultante
*/
public static void showResultSet(Statement stat) throws SQLException
{
ResultSet result = stat.getResultSet();
ResultSetMetaData metaData = result.getMetaData(); int columnCount = metaData.getColumnCount();
for (int i = 1; i <= columnCount; i++)
{
if (i > 1) System.out.print(", ");
System.out.print(metaData.getColumnLabel(i));
}
System.out.println();
while (result.next())
{
for (int i = 1; i <= columnCount; i++)
{
if (i > 1) System.out.print(", ");
System.out.print(metaData.getString(i));
}
System.out.println();
}
result.close();
}
}
```

5

Caso di studio: la base di dati di una banca

In questo paragrafo svilupperemo un programma completo per una base di dati: la simulazione di uno sportello bancario automatico (ATM, Automatic Teller Machine) che memorizza in una base di dati i dati relativi ai clienti e ai loro conti bancari. In

tale simulazione, ciascun cliente ha un numero cliente che lo caratterizza, un PIN (Personal Identification Number) e due conti bancari (un conto corrente e un conto di risparmio). Memorizzeremo le informazioni in due tabelle: ***Customer***

Customer_Number

PIN

Checking_Account_Number

Savings_Account_Number

INTEGER

INTEGER

INTEGER

INTEGER

Account

Account_Number

Balance

INTEGER

DECIMAL(10, 2)

La classe Bank si deve connettere alla base di dati ogni volta che deve riconoscere un cliente; ecco l'implementazione del metodo che cerca un cliente, eseguendo l'interrogazione seguente:

*SELECT * FROM Customer WHERE Customer_Number = ...*

Successivamente, verifica che il PIN sia corretto, e costruisce un oggetto di tipo Customer. Questo metodo trasforma in dati orientati agli oggetti le informazioni della base di dati, che sono organizzate in righe e colonne.

```
public Customer find(int customerNumber, int pin) throws SQLException
{
Customer c = null;
Connection conn = SimpleDataSource.getConnection();
Statement stat = conn.createStatement();
ResultSet result = stat.executeQuery("SELECT *"
+ " FROM Customer WHERE Customer_Number = "
+ customerNumber);
```

if (result.next() && pin == result.getInt("PIN")) c = new Customer(customerNumber,
result.getInt("Checking_Account_Number"),
result.getInt("Savings_Account_Number"));
result.close();
stat.close();
conn.close();
return c;
}

Notate che il metodo lancia un'eccezione di tipo SQLException: perché non la cattu-riamo, restituendo null? Ci sono molti motivi che possono sollevare un'eccezione SQL e la classe Bank non vuole nasconderne i dettagli. Ma la classe Bank, al tempo stesso, non sa come sia strutturata l'interfaccia utente dell'applicazione, per cui non può visualizzare le informazioni relative all'eccezione in modo che l'utente le possa vedere. Rinviando l'eccezione al metodo chiamante, l'informazione può raggiungere la sezione del programma che interagisce con l'utente.

La classe BankAccount di questo programma è abbastanza diversa da quella che abbiamo implementato negli altri capitoli del libro, perché ora non dobbiamo memorizzare il saldo di un conto bancario nell'oggetto, ma lo cerchiamo nella base di dati:

public double getBalance()
throws SQLException
{
double balance = 0;

Connessione a base di dati
Connection conn = SimpleDataSource.getConnection(); Statement stat = conn.createStatement();
ResultSet result = stat.executeQuery("SELECT Balance"
+ " FROM Account WHERE Account_Number = "
+ accountNumber);
if (result.next())
balance = result.getDouble(1);
result.close();
stat.close();

```
conn.close();
return balance;
}
```

Allo stesso modo, le operazioni deposit e withdraw aggiornano immediatamente la base di dati:

```
public void deposit(double amount)
throws SQLException
{
Connection conn = SimpleDataSource.getConnection();
Statement stat = conn.createStatement();
stat.execute("UPDATE Account"
+ " SET Balance = Balance + " + amount
+ " WHERE Account_Number = " + accountNumber);
stat.close();
conn.close();
}
```

Sembra piuttosto inefficiente connettersi alla base di dati ogni volta che si accede al saldo di un conto, ma ciò è molto più sicuro della sua memorizzazione in un oggetto.

Supponete di avere due copie del programma ATM in esecuzione nello stesso momento: in questo caso, è possibile che entrambi i programmi modifichino lo stesso conto corrente. Se ciascuno di essi avesse copiato all'interno di oggetti i saldi dei conti bancari contenuti nella base di dati, le modifiche fatte da un utente non verrebbero viste dall'altro.

Potete provare anche voi a compiere questo accesso simultaneo, eseguendo semplicemente due copie della simulazione di ATM. In alternativa, potete modificare il metodo main della classe ATMSimulation in modo che visualizzi due finestre frame di tipo ATM, ma avrete bisogno di una base di dati "di potenza industriale", perché alcune basi di dati non permettono connessioni simultanee da programmi diversi.

Evidentemente, l'accesso contemporaneo da parte di più utenti è, in pratica, assai importante: la maggior parte delle vere applicazioni per basi di dati hanno molti utenti simultaneamente, per cui dovete progettare con molta cura il trasporto delle informazioni dalla base di dati a oggetti del programma. Non potete semplicemente copiare i dati in oggetti

che l'utente possa modificare; d'altra parte, le ricerche nella base di dati sono piuttosto lente, se comparate alle ricerche in oggetti, per cui dovreste conservare i dati immutabili in oggetti. Programmare applicazioni realistiche per basi di dati che vengano eseguite velocemente e correttamente può diventare abbastanza complicato. Negli ultimi anni, si sono resi disponibili dei server di applicazioni che gestiscono per conto degli sviluppatori alcuni di questi problemi. Un server di appli-

cazioni tiene sotto controllo gli oggetti che copiano informazioni da una base di dati e si accertano che i dati negli oggetti e nella base di dati siano sempre sincronizzati. Una tecnologia standard per server di applicazioni basati su Java è definita in Java 2 Enter-prise Edition.

File ATMSimulation.java

```
import javax.swing.JFrame;
/**
Simulazione di uno sportello bancario automatico.
*/
public class ATMSimulation
{
public static void main(String[] args)
{
if (args.length == 0)
{
System.out.println(
"Usage: java ATMSimulation propertiesFile");
System.exit(0);
}
else
{
try
{
SimpleDataSource.init(args[0]);
}
catch (Exception ex)
{
```

```
    ex.printStackTrace();
    System.exit(0);
    }
    }
    JFrame frame = new ATM();
    frame.setTitle("First National Bank of Java");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
    frame.show();
    }
    }
```

File ATM.java

```
    import java.awt.Container;
    import java.awt.FlowLayout;
    import java.awt.GridLayout;
    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
    import java.io.FileInputStream;
    import java.io.IOException;
    import java.sql.SQLException;
```



```
    import java.util.Properties;
    import javax.swing.JButton;
    import javax.swing.JFrame;
    import javax.swing.JOptionPane;
    import javax.swing.JPanel;
    import javax.swing.JTextArea;
    /**
    Un frame che visualizza i componenti dell'ATM.
    */
    class ATM extends JFrame
    {
    /**
    Costruisce l'interfaccia utente dell'applicazione ATM.
    */
    public ATM()
    {
```

theBank = new Bank();
// costruisci i componenti
pad = new KeyPad();
display = new JTextArea(4, 20);
aButton = new JButton(“ A “);
aButton.addActionListener(new AButtonListener()); bButton = new JButton(“ B “);
bButton.addActionListener(new BButtonListener()); cButton = new JButton(“ C “);
cButton.addActionListener(new CButtonListener());
// aggiungi i componenti al pannello dei contenuti JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(3, 1));
buttonPanel.add(aButton);
buttonPanel.add(bButton);
buttonPanel.add(cButton);
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
contentPane.add(pad);
contentPane.add(display);
contentPane.add(buttonPanel);
try
{
setNextState(START_STATE);
}
catch (SQLException exception)
{
Connessione a base di dati
37
JOptionPane.showMessageDialog(null,
exception.getMessage());
}
}
*/**
Imposta il numero cliente attuale al valore digitato sul tastierino e imposta lo stato a PIN.*
/
public void setCustomerNumber() throws SQLException

```
{
customerNumber = (int)pad.getValue();
setNextState(PIN_STATE);
}
/**
Legge il PIN dal tastierino e cerca il cliente nella banca.
Se lo trova imposta lo stato a ACCOUNT, altrimenti a START.
*/
public void selectCustomer() throws SQLException
{
int pin = (int)pad.getValue();
currentCustomer = theBank.find(
customerNumber, pin);
if (currentCustomer == null)
setNextState(START_STATE);
else
setNextState(ACCOUNT_STATE);
}
/**
Imposta il conto attuale al conto corrente o di risparmio.
Imposta lo stato a TRANSACT.
@param account CHECKING_ACCOUNT oppure
SAVINGS_ACCOUNT
*/
public void selectAccount(int account)
throws SQLException
{
if (account == CHECKING_ACCOUNT)
currentAccount =
currentCustomer.getCheckingAccount();
else
currentAccount =
currentCustomer.getSavingsAccount();
setNextState(TRANSACT_STATE);
}
/**
Preleva dal conto attuale l'importo digitato nel tastierino.
Imposta lo stato a ACCOUNT.
```

```
*/
public void withdraw() throws SQLException
{
currentAccount.withdraw(pad.getValue());
```

```
setNextState(ACCOUNT_STATE);
}
/**
Versa nel conto attuale l'importo digitato nel tastierino.
Imposta lo stato a ACCOUNT.
*/
public void deposit() throws SQLException
{
currentAccount.deposit(pad.getValue());
setNextState(ACCOUNT_STATE);
}
/**
Imposta lo stato e aggiorna il messaggio visualizzato.
@param newState il nuovo stato
*/
public void setNextState(int newState)
throws SQLException
{
state = newState;
pad.clear();
if (state == START_STATE)
display.setText(
"Enter customer number\nA = OK");
else if (state == PIN_STATE)
display.setText("Enter PIN\nA = OK");
else if (state == ACCOUNT_STATE)
display.setText("Select Account\n"
+ "A = Checking\nB = Savings\nC = Exit");
else if (state == TRANSACT_STATE)
display.setText("Balance = "
+ currentAccount.getBalance()
+ "\nEnter amount and select transaction\n"
```

```
+ "A = Withdraw\nB = Deposit\nC = Cancel");
}
private class AButtonListener implements ActionListener
{
public void actionPerformed(ActionEvent event)
{
try
{
if (state == START_STATE)
setCustomerNumber();
else if (state == PIN_STATE)
selectCustomer();
else if (state == ACCOUNT_STATE)
selectAccount(CHECKING_ACCOUNT);
else if (state == TRANSACT_STATE)
withdraw();
}
catch (SQLException exception)
{
```



```
JOptionPane.showMessageDialog(null,
exception.getMessage());
}
}
}
private class BButtonListener implements ActionListener
{
public void actionPerformed(ActionEvent event)
{
try
{
if (state == ACCOUNT_STATE)
selectAccount(SAVINGS_ACCOUNT);
else if (state == TRANSACT_STATE)
deposit();
}
catch (SQLException exception)
```

```
{
JOptionPane.showMessageDialog(null,
exception.getMessage());
}
}
}
private class CButtonListener implements ActionListener
{
public void actionPerformed(ActionEvent event)
{
try
{
if (state == ACCOUNT_STATE)
setNextState(START_STATE);
else if (state == TRANSACT_STATE)
setNextState(ACCOUNT_STATE);
}
catch (SQLException exception)
{
JOptionPane.showMessageDialog(null,
exception.getMessage());
}
}
}
private int state;
private int customerNumber;
private Customer currentCustomer;
private BankAccount currentAccount;
private Bank theBank;
private JButton aButton;
private JButton bButton;
private JButton cButton;
```



```
private KeyPad pad;
private JTextArea display;
private static final int START_STATE = 1;
private static final int PIN_STATE = 2;
```

```
private static final int ACCOUNT_STATE = 3;
private static final int TRANSACT_STATE = 4;
private static final int CHECKING_ACCOUNT = 1;
private static final int SAVINGS_ACCOUNT = 2;
}
```

File KeyPad.java

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JTextField;
/**
Un componente che consente all'utente di digitare un numero, usando un tastierino con pulsanti etichettati con cifre.
*/
public class KeyPad extends JPanel
{
/**
Costruisce il pannello del tastierino.
*/
public KeyPad()
{
setLayout(new BorderLayout());
// aggiungi il campo di visualizzazione
display = new JTextField();
add(display, BorderLayout.NORTH);
// costruisci il pannello dei pulsanti
buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(4, 3));
// aggiungi i pulsanti per le cifre
addButton("7");
addButton("8");
addButton("9");
addButton("4");
addButton("5");
addButton("6");
```

```
addButton("1");
addButton("2");
addButton("3");
addButton("0");
addButton(".");
// aggiungi il pulsante di cancellazione
clearButton = new JButton("CE");
buttonPanel.add(clearButton);
class ClearButtonListener implements ActionListener
{
public void actionPerformed(ActionEvent event)
{
display.setText("");
}
}
clearButton.addActionListener(
new ClearButtonListener());
add(buttonPanel, BorderLayout.CENTER);
}
/**
Aggiunge un pulsante al pannello dei pulsanti.
@param label l'etichetta del pulsante
*/
private void addButton(final String label)
{
class DigitButtonListener implements ActionListener
{
public void actionPerformed(ActionEvent event)
{
// non aggiungere due punti decimali
if (label.equals(".")
&& display.getText().indexOf(".") != -1) return;
display.setText(display.getText() + label);
}
}
JButton button = new JButton(label);
```

```
buttonPanel.add(button);
button.addActionListener(new DigitButtonListener());
}
/**
Fornisce il valore digitato dall'utente.
@return il valore presente nel campo di testo
del tastierino
*/
```



```
public double getValue()
{
return Double.parseDouble(display.getText());
}
/**
Azzera il visualizzatore.
*/
public void clear()
{
display.setText();
}
private JPanel buttonPanel;
private JButton clearButton;
private JTextField display;
}
```

File Bank.java

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;
/**
Una banca contiene clienti con i loro conti bancari.
*/
public class Bank
{
/**
Cerca nella banca un cliente avente un dato numero cliente e PIN.
```

@param customerNumber un numero cliente
@param pin un numero di identificazione personale
@return il cliente corrispondente
oppure null se nessun cliente corrisponde
**/*
public Customer find(int customerNumber, int pin) throws SQLException
{
Customer c = null;
Connection conn = SimpleDataSource.getConnection(); Statement stat = conn.createStatement();
*ResultSet result = stat.executeQuery("SELECT *"*
+ " FROM Customer WHERE Customer_Number = "
+ customerNumber);
if (result.next() && pin == result.getInt("PIN")) c = new Customer(customerNumber,
result.getInt("Checking_Account_Number"),
result.getInt("Savings_Account_Number"));
result.close();
stat.close();
Connessione a base di dati
43
conn.close();
return c;
}
}
File BankAccount.java
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;
*/***
Un conto bancario ha un saldo che può essere modificato con versamenti e prelievi.
**/*
public class BankAccount
{
*/***

Costruisce un conto bancario con un numero di conto assegnato.

```
@param anAccountNumber il numero di conto
*/
public BankAccount(int anAccountNumber)
{
accountNumber = anAccountNumber;
}
/**
Versa denaro nel conto bancario.
@param amount la somma da versare
*/
public void deposit(double amount)
throws SQLException
{
Connection conn = SimpleDataSource.getConnection();
Statement stat = conn.createStatement();
stat.execute("UPDATE Account"
+ " SET Balance = Balance + " + amount
+ " WHERE Account_Number = " + accountNumber);
stat.close();
conn.close();
}
/**
Preleva denaro dal conto bancario.
@param amount la somma da prelevare
*/
public void withdraw(double amount)
throws SQLException
{
Connection conn = SimpleDataSource.getConnection();
Statement stat = conn.createStatement();
stat.execute("UPDATE Account"
+ " SET Balance = Balance - " + amount
```



```
+ " WHERE Account_Number = " + accountNumber);
stat.close();
```

```
conn.close();
}
/**
Restituisce il saldo del conto bancario.
@return il saldo del conto
*/
public double getBalance()
throws SQLException
{
double balance = 0;
Connection conn = SimpleDataSource.getConnection();
Statement stat = conn.createStatement();
ResultSet result = stat.executeQuery(
"SELECT Balance"
+ " FROM Account WHERE Account_Number = "
+ accountNumber);
if (result.next())
balance = result.getDouble(1);
result.close();
stat.close();
conn.close();
return balance;
}
private int accountNumber;
}
```

File Customer.java

```
/**
Un cliente della banca con un conto corrente e un conto di risparmio.
*/
public class Customer
{
/**
Costruisce un cliente con un numero cliente, un numero di conto corrente e un numero di conto di risparmio assegnati.
@param aCustomerNumber il numero cliente
@param checkingAccountNumber il numero del conto corrente
```

@param savingsAccountNumber il numero del conto di risparmio
**/*
public Customer(int aCustomerNumber,
int checkingAccountNumber,
int savingsAccountNumber)
{
customerNumber = aCustomerNumber;
checkingAccount = new BankAccount(
checkingAccountNumber);
savingsAccount = new BankAccount(
savingsAccountNumber);
}
Connessione a base di dati
45
*/***
Fornisce il conto corrente del cliente.
@return il conto corrente
**/*
public BankAccount getCheckingAccount()
{
return checkingAccount;
}
*/***
Fornisce il conto di risparmio del cliente.
@return il conto di risparmio
**/*
public BankAccount getSavingsAccount()
{
return savingsAccount;
}
private int customerNumber;
private BankAccount checkingAccount;
private BankAccount savingsAccount;
}
6
Concetti avanzati per le basi di dati

In questo paragrafo accenneremo brevemente ad alcuni concetti più avanzati relativi alle basi di dati, che rivestono grande importanza nelle applicazioni reali. Se seguirete un corso di programmazione di basi di dati, affronterete di nuovo questi argomenti in maggiore dettaglio, mentre a questo punto dei vostri studi potreste dare un'occhiata a questo paragrafo per capire meglio che la programmazione di basi di dati reali è pa-recchio più complessa dell'invio di una manciata di interrogazioni SQL.

6.1

Transazioni

Una transazione è un insieme di

Una parte importante dell'elaborazione delle basi di dati è la gestione aggiornamenti di una base di dati

delle transazioni: una transazione è un insieme di aggiornamenti di una che devono essere eseguiti tutti

base di dati che devono essere eseguiti tutti completamente oppure non completamente oppure non essere eseguiti per nulla.

essere eseguiti per nulla. Ad esempio, considerate un'applicazione bancaria che trasferisce denaro da un conto a un altro. Questa operazione è costituita da due passi: il saldo di un conto viene diminuito e il saldo di un altro conto viene aumentato. Nessun sistema software è perfetto e c'è la possibilità che si verifichi un errore. L'applicazione bancaria, il programma della base di dati o la connessione di rete fra loro potrebbe generare un errore subito dopo l'esecuzione del primo passo, per cui il denaro sarebbe stato prelevato dal primo conto senza essere stato depositato nel secondo conto. Ovviamente, questo sarebbe un grave errore, ed esistono molte altre situazioni simili. Per esempio, se modificate una prenotazione per un volo aereo, non volete rendere disponibile il vostro vecchio posto prima che il vostro nuovo posto sia stato confermato.

Ciò che tutte queste situazioni hanno in comune è che un insieme di operazioni di una base di dati sono raggruppate per

costituire una transazione. Tutte le operazioni del gruppo devono essere portate a termine completamente: un'esecuzione parziale non è ammessa. In SQL, si usano i comandi COMMIT e ROLLBACK per gestire le transazioni. Ad esempio, per trasferire denaro da un conto a un altro, inviate questi comandi:

UPDATE Account SET Balance = Balance - 1000

WHERE Account_Number = ‘95667-2574’
UPDATE Account SET Balance = Balance + 1000
WHERE Account_Number = ‘82041-1196’
COMMIT

Il comando COMMIT rende definitivi gli aggiornamenti. Al contrario, il comando ROLLBACK annulla tutte le modifiche fino all’ultimo COMMIT che è stato eseguito.

Quando programmate con la tecnologia JDBC, l’impostazione predefinita della libreria JDBC prevede l’esecuzione definitiva di tutti gli aggiornamenti alla base di dati. Questo è comodo per programmi semplici, ma non è ciò che si vuole per la gestione delle transazioni, per cui dovete, prima di tutto, disattivare la modalità automatica di aggiornamento definitivo:

Connection conn = ...;
conn.setAutoCommit(false);
Statement stat = conn.getStatement();

Quindi, inviate gli aggiornamenti che costituiscono la transazione, e invocate il metodo commit della classe Statement.

stat.esecuteUpdate(
“UPDATE Account SET Balance = Balance - “
+ amount + “ WHERE Account_Number = “ + fromAccount);
stat.esecuteUpdate(
“UPDATE Account SET Balance = Balance + “
+ amount + “ WHERE Account_Number = “ + toAccount);
stat.commit();

Al contrario, se si verifica un errore, chiamate il metodo rollback. Questa situazione si verifica, tipicamente, in un gestore di eccezioni: try

{
...
}
catch (SQLException exception)
{
stat.rollback();
}

Vi potreste chiedere come faccia una base di dati ad annullare aggiornamenti già eseguiti quando ciò viene richiesto.

La base di dati, in realtà, memorizza le modifiche in un insieme di tabelle temporanee: se effettuare interrogazioni all'interno di una tran-

	INVOICE_NUMBER	PRODUCT_CODE	QUANTITY
0	11731	116-064	3
1	11731	643-119	2
2	11731	257-535	1
3	11732	116-064	10
4	11733	116-064	2

sazione, le informazioni presenti nelle tabelle temporanee vengono fuse con i dati permanenti allo scopo di calcolare il risultato dell'interrogazione, fornendovi l'illusione che gli aggiornamenti abbiano già avuto luogo. Quando rendete definitiva la transazione, i dati temporanei vengono resi permanenti. Quando richiedete l'annulla-mento della transazione, le tabelle temporanee vengono semplicemente eliminate.

6.2 Insiemi di risultati scorrevoli e aggiornabili

Quando ottenete un insieme di risultati da un'interrogazione, potete esaminarne il contenuto una riga per volta, spostandovi in avanti con il metodo next. Tuttavia, può darsi che vogliate anche tornare indietro: ciò accade molto spesso quando i risultati vengono visualizzati in un componente dell'interfaccia utente grafica come una tabella dotata di barre di scorrimento (osservate la Figura 12). Può darsi che l'utente voglia scorrere all'indietro per osservare di nuovo alcuni dati.

Ovviamente, se l'insieme dei risultati è piccolo, è semplice copiarlo in un array, ma i risultati delle interrogazioni possono essere enormi. Ad esempio, supponete che un'interrogazione abbia 100000 possibili risultati. (La vostra personale esperienza vi insegna che alcune interrogazioni fatte a un motore di ricerca sul Web generano un numero di risultati enorme.) In questo caso, non è pratico leggere l'intero insieme e copiarlo in un array. Ricordate, anche, che la base di dati potrebbe trovarsi in un altro computer, per cui il trasferimento di grandi quantità di

dati dalla base di dati alla vostra applicazione potrebbe essere un'operazione piuttosto lenta. Per questa ragione, l'oggetto di tipo ResultSet non contiene l'intero insieme dei risultati: è soltanto uno strumento di collegamento con l'insieme dei risultati che è memorizzato nella base di dati. Ogni volta che invocate il metodo next, una singola riga viene trasferita nell'oggetto di tipo ResultSet.

Indipendentemente dalla velocità, è probabile che un utente non abbia bisogno di guardare tutti i risultati di una grande interrogazione. Tipicamente, l'utente osserva le prime righe del risultato e poi cerca di capire come restringere la ricerca. Anche in questo caso, esaminando i risultati, gli utenti possono voler tornare indietro. Per rendere semplice la programmazione di tutto questo, potete usare un insieme di risultati scorrevole (scrollable).

Per ottenere insiemi di risultati scorrevoli, dovete creare un oggetto di tipo Statement che usi alcune costanti che si trovano nella classe ResultSet e che rappresentano tipi e parametri di concorrenza. Ecco le possibili alternative:

■ Volete un insieme di risultati scorrevole? Se non lo volete, usate TYPE_FORWARD_ONLY, il tipo predefinito.

Figura 12

Una vista scorrevole

■ Mentre scorrete un insieme di risultati, volete che l'insieme venga aggiornato mentre altri processi modificano le informazioni presenti nella base di dati? In questo caso, usate TYPE_SCROLL_SENSITIVE, altrimenti TYPE_SCROLL_INSENSITIVE.

■ Volete essere in grado di aggiornare la base di dati modificando l'insieme di risultati (invece di inviare comandi di tipo U P D A T E)? In questo caso, usate CONCUR_UPDATABLE, altrimenti CONCUR_READ_ONLY. Parleremo degli insiemi di risultati aggiornabili più avanti in questo paragrafo.

Ad esempio, supponiamo che vogliate effettuare scorrimenti, ma che non vi interessi che il risultato sia sensibile alle modifiche eventualmente apportate da altri processi, e che non

dobbiate aggiornare l'insieme dei risultati. Di conseguenza, costruite l'oggetto di tipo Statement come segue:

Statement stat = conn.createStatement(

ResultSet.TYPE_SCROLL_INSENSITIVE,

ResultSet.CONCUR_READ_ONLY);

ResultSet result = stat.executeQuery(...);

Una volta che avete ottenuto da un'interrogazione un insieme di risultati scorrevole, avete parecchi metodi per navigare attraverso i risultati:

- *previous() torna alla riga precedente.*
- *first() e last() si posiziona, rispettivamente, sulla prima e sull'ultima riga.*
- *relative(n) e absolute(n) si sposta di n righe, oppure si posiziona sulla riga numero n.*

Un insieme di risultati scorrevo-

In effetti, un insieme di risultati scorrevole vi fornisce un accesso casuale consente un accesso casuale

le all'insieme stesso.

all'insieme stesso.

Se disponete di un insieme di risultati aggiornabile, potete usare alcuni metodi per aggiornare i valori corrispondenti nella base di dati. Ad esempio, supponete di avere una riga che rappresenta un conto bancario.

Statement stat = conn.createStatement(

ResultSet.TYPE_FORWARD_ONLY,

ResultSet.CONCUR_UPDATABLE);

String accountQuery =

*"SELECT * WHERE Account_Number = " + account;*

ResultSet result = stat.executeQuery(accountQuery); double balance = result.getDouble("Balance");

Quindi, potete invocare

result.updateDouble("Balance", balance + deposit); Questa chiamata aggiorna la riga attuale ma non ha ancora aggiornato la base di dati.

Potete aggiornare altri campi nella stessa riga, poi invocare result.updateRow();

per modificare i dati nella base di dati. Potete anche usare insiemi di risultati aggiornabili per inserire nuove righe: per

prima cosa spostate il cursore in una posizione Connessione a base di dati

49

speciale, detta riga di inserimento, poi aggiornate i valori di tutte le colonne e invocate il metodo updateRow:

result.moveToInsertRow();

result.updateString("Account_Number", "54382-5516"); result.updateString("Customer_Number", "992345"); result.updateDouble("Balance", 1000);

result.updateRow();

Molti programmatori ritengono che gli insiemi di risultati aggiornabili siano più semplici da utilizzare dei comandi SQL UPDATE o INSERT.

Un insieme di risultati aggiorna-

Non tutti i driver JDBC implementano gli insiemi di risultati scorre-bile rende più semplice la modi-

voli o aggiornabili. Se usate un driver che non ha queste potenzialità, fica del contenuto di una base di

l'insieme di risultati che ottenete da un'interrogazione non avrà la pos-dati, senza usare comandi SQL.

sibilità di essere scorso o aggiornato: quando invocate uno dei metodi non funzionanti (come il metodo previous o update), viene lanciata un'eccezione.

6.3

Velocizzare le interrogazioni

Esistono alcuni metodi per velocizzare le interrogazioni. In questo paragrafo ne presenteremo tre:

- *Indicizzazione*
- *Enunciati predisposti*
- *Procedure memorizzate*

Gli indici di una tabella possono

Ricordate che una chiave primaria è una colonna (o una combinazione velocizzare le interrogazioni alla

di colonne) che identifica univocamente una riga in una tabella. Quan-base di dati. Per usare gli indici,

do una tabella ha una chiave primaria, la base di dati può costruire un dovete specificare le chiavi primarie quando costruite le tabelle.

file di indici: un file che contiene informazioni su come accedere velocemente a una riga quando si conosce la sua chiave primaria. L'indicizzazione può rendere notevolmente più veloci le operazioni di unione.

Se la chiave primaria fa parte di una sola colonna, potete contrassegnare tale colonna con la parola chiave PRIMARY KEY, in questo modo: CREATE TABLE Product

(

Product_Code CHAR(10) PRIMARY KEY,

Description CHAR(40),

Price DECIMAL(10, 2)

)

Se la chiave primaria è una combinazione di più colonne, aggiungete una clausola PRIMARY KEY alla fine del comando CREATE TABLE, in questo modo: CREATE TABLE Item

(

Invoice_Number INTEGER,

Product_Code CHAR(10),

Quantity INTEGER,

PRIMARY KEY (Invoice_Number, Product_Code)

)

A volte è possibile velocizzare le interrogazioni costruendo indici secondari: file di indici che indicizzano altri insiemi di colonne, senza che siano necessariamente univoci.

Si tratta di una tecnica avanzata di cui non parleremo.

Usate enunciati predisposti per

Quando inviate un'interrogazione a una base di dati, il programma interrogazioni e aggiornamenti

che la gestisce deve capire in quale ordine eseguire la ricerca nelle ta-che vengono ripetuti di frequen-

belle che volete interrogare. Identificare una strategia per rispondere a te, per evitare la ripetuta identificazione di strategie di risposta.

una interrogazione richiede tempo: se inviate molte volte la stessa interrogazione, ha senso usare un enunciato predisposto (prepared statement).

Quando inviate a una base di dati un enunciato predisposto, la base di dati formula una strategia di interrogazione e la memorizza all'interno dell'enunciato stesso, predisponendolo per successive interrogazioni: ogni volta che inviate l'enunciato predisposto, la base di dati utilizza quella strategia precalcolata. Quindi, usate enunciati predisposti quando dovete inviare ripetutamente gli stessi enunciati.

*Ad esempio, ecco come predisporre l'interrogazione che cerca le informazioni relative al conto bancario avente un certo numero di conto: String query = "SELECT * WHERE Account_Number = ?"; PreparedStatement prepStat = conn.prepareStatement(query); Il simbolo ? nella stringa di interrogazione indica le variabili di cui fornirete il valore quando eseguirete realmente l'interrogazione, usando un metodo di tipo set, in questo modo:*

prepStat.setString(1, accountNumber);

Il primo parametro dei metodi set indica la posizione della variabile: 1 per il primo ?, 2 per il secondo, e così via.

Infine, quando avete impostato tutte le variabili, chiamate il metodo executeQuery, come fareste con un normale enunciato.

ResultSet result = prepStat.executeQuery();

Potete continuare a riutilizzare lo stesso oggetto che rappresenta un'interrogazione predisposta: semplicemente, impostate nuovi valori per le variabili e invocate executeQuery.

Le procedure memorizzate sono un altro metodo per formulare enunciati ripetuti. Le procedure memorizzate vengono eseguite nel nucleo (kernel) della base di dati e sono quindi molto più veloci delle sequenze di comandi SQL, ciascuno dei quali viene trasmesso individualmente alla base di dati. Non tutte le basi di dati consentono di eseguire procedure memorizzate, e quelle che lo fanno non usano tutte la stessa sintassi.

(Alcune basi di dati, che aderiscono allo standard SQLJ, vi consentono addirittura di formulare procedure memorizzate in Java! Tuttavia, non parleremo del linguaggio SQLJ.) Di conseguenza, prima di provare a eseguire il codice che segue, dovete consultare la documentazione della vostra base di dati.

Connessione a base di dati

Le procedure memorizzate ven-

Prima che le possiate invocare, dovete inserire le procedure memo-gono eseguite nel nucleo (kernel)

rizzate nella base di dati, cosa che tipicamente si fa inviando un coman-della base di dati e sono quindi

do CREATE PROCEDURE.

molto più veloci delle sequenze

Ecco un esempio di procedura che trasferisce denaro da un conto di comandi SQL.

bancario a un altro:

CREATE PROCEDURE transfer(

:from INTEGER, :to INTEGER, :amount DECIMAL(10, 2)) BEGIN

UPDATE Account SET Balance = Balance - :amount

WHERE Account_Number = :from;

UPDATE Account SET Balance = Balance + :amount

WHERE Account_Number = :from;

END

In Java, potete inserire nella base di dati la procedura memorizzata eseguendo l'enunciato SQL che la definisce.

```
String definition =
"CREATE PROCEDURE transfer("
+ ...
+ ") BEGIN"
+ ...
+ "END";
stat.execute(definition);
```

Per invocare la procedura, usate la classe CallableStatement; si può ottenere un oggetto di tipo CallableStatement usando il metodo prepareCall della classe Connection.

```
CallableStatement callStat = conn.prepareCall(
"{call transfer(?, ?, ?)}");
```

I caratteri ? che fungono da segnaposto hanno la stessa funzione che avevano negli enunciati predisposti: dovete impostarne il valore prima di invocare la procedura. Le parentesi { } indicano al driver JDBC che il comando call racchiuso all'interno non è un comando SQL standard e che

deve essere tradotto nella sintassi di invocazione di quella particolare base di dati.

Successivamente, impostate i valori dei parametri: callStat.setInt(1, fromAccount);

callStat.setInt(2, toAccount);

callStat.setDouble(3, amount);

Infine, eseguite l'invocazione:

callStat.execute();

L'uso di procedure memorizzate per sequenze di enunciati SQL inviate frequentemente può migliorare in modo significativo le prestazioni di un'applicazione di basi di dati, ma, poiché le procedure memorizzate differiscono da una base di dati all'altra, il ***52***

Connessione a base di dati

loro utilizzo limita la portabilità. La cosa migliore è far funzionare correttamente l'applicazione e caratterizzarne le prestazioni, prima di ricorrere alle procedure memorizzate.

In questo paragrafo avete visto alcune tecniche avanzate per la programmazione di basi di dati. Nonostante la logica di funzionamento della maggior parte delle applicazioni di basi di dati possa essere espressa con i soli comandi basilari presentati nella prima parte del capitolo, i programmatori di basi di dati professionisti usano queste e altre tecniche avanzate per generare applicazioni con buone prestazioni.

La discussione di queste tecniche avanzate termina questo capitolo sulla programmazione di basi di dati in Java. Avete visto come usare il linguaggio SQL per interrogare e aggiornare i dati in una base di dati e come la libreria JDBC semplifichi l'invio di comandi SQL dall'interno di un programma scritto in Java.

Riepilogo del capitolo

1. Una base di dati (database) relazionale memorizza le informazioni all'interno di tabelle. Ciascuna colonna di una tabella è caratterizzata da un nome e da un tipo di dati.

2. Il linguaggio SQL (Structured Query Language) è un linguaggio imperativo per interagire con una base di dati.

3. Usate i comandi SQL CREATE TABLE e INSERT per aggiungere dati a una base di dati.

4. Dovreste evitare righe con dati replicati. Distribuite, invece, i dati su più tabelle.

5. Una chiave primaria è una colonna (o una combinazione di colonne) i cui valori identificano univocamente una riga di una tabella.

6. Una chiave esterna è un riferimento a una chiave primaria in una tabella collegata.

7. Per interrogare una base di dati, usate l'enunciato SELECT.

8. Una unione è un'interrogazione che coinvolge più tabelle.

9. I comandi UPDATE e DELETE modificano i dati nella base di dati.

10. Per accedere a una base di dati da un programma Java avete bisogno di un driver JDBC (Java Database Connectivity).

11. Accertatevi che il driver JDBC si trovi nel percorso delle classi (classpath) prima di eseguire il programma Java.

12. Per accedere a una base di dati da un programma Java usate un oggetto di tipo Connection.

13. Un oggetto di tipo Connection può creare oggetti di tipo Statement, che si usano per eseguire comandi SQL.

14. Il risultato di un'interrogazione SQL viene restituito in un oggetto di tipo ResultSet.

15. I dati descrittivi (meta data) si riferiscono a un oggetto. I dati descrittivi di un insieme di risultati descrivono le proprietà dell'insieme stesso.

Connessione a base di dati

53

16. Una transazione è un insieme di aggiornamenti di una base di dati che devono essere eseguiti tutti completamente oppure non essere eseguiti per nulla.

17. Un insieme di risultati scorrevole consente un accesso casuale all'insieme stesso.

18. Un insieme di risultati aggiornabile rende più semplice la modifica del contenuto di una base di dati, senza usare comandi SQL.

19. Gli indici di una tabella possono velocizzare le interrogazioni alla base di dati. Per usare gli indici, dovete specificare le chiavi primarie quando costruite le tabelle.

20. Usate enunciati predisposti per interrogazioni e aggiornamenti che vengono ripetuti di frequente, per evitare la ripetuta identificazione di strategie di risposta.

21. Le procedure memorizzate vengono eseguite nel nucleo (kernel) della base di dati e sono quindi molto più veloci delle sequenze di comandi SQL.

Ulteriori letture

[1] Chris J. Date e Hugh Darwen, A Guide to the SQL Standard: A User's Guide to the Standard Database Language SQL, Addison-Wesley, 1997.

[2] Seth White, Maydene Fisher, Rick Cattell, Graham Hamilton e Mark Hap-ner, JDBC API Tutorial and Reference, 2nd edition, Addison-Wesley, 1999.

[3] http://www.cpsr.org/cpsr/privacy/ssn/ssn.faq.html Un elenco di risposte a domande ricorrenti sui numeri della previdenza sociale statunitense, gestito da Computer Professionals for Social Responsibility.

[4] http://www.ftc.gov/privacy/ Un sito Web statunitense con informazioni sulla privacy gestito da Federal Trade Commission.

Classi, oggetti e metodi
presentati nel capitolo

java.lang.Class
forName
java.sql.Connection
close
commit
createStatement
prepareCall
prepareStatement
rollback
setAutoCommit
java.sql.DriverManager
getConnection
java.sql.PreparedStatement
execute
executeQuery
executeUpdate
setInt

setDouble
setString
java.sql.ResultSet
absolute
close
first
getInt
getDouble
getMetaData
getString
last
moveToInsertRow
next
previous
relative
updateInt
updateDouble
updateRow
updateString
java.sql.ResultSetMetaData
getColumnCount
getColumnDisplaySize
getColumnLabel
java.sql.SQLException
java.sql.Statement
close
execute
executeQuery
executeUpdate
getResultSet
getUpdateCount
java.util.Properties
load
getProperty
Esercizi di ripasso

Esercizio R.1. *Progettate un insieme di tabelle per una base di dati in grado di memorizzare persone e automobili. Una persona ha un nome, un numero di patente di guida univoco e un indirizzo. Ciascuna automobile è caratterizzata da: un numero di identificazione del veicolo univoco; il nome del costruttore; il nome del modello; l'an-no. Ciascuna automobile ha un proprietario, ma una persona può possedere più automobili.*

Esercizio R.2. *Progettate un insieme di tabelle per una base di dati in grado di memorizzare i libri di una biblioteca e i lettori che la frequentano. Un libro ha un codice ISBN (International Standard Book Number), un autore e un titolo. La biblioteca può avere più copie dello stesso libro, ciascuna con un diverso codice ID del libro. Un lettore ha un nome, un ID identificativo univoco e un indirizzo. Un libro può essere preso a prestito da un solo lettore per volta, ma un lettore può prendere a prestito più libri contemporaneamente.*

Esercizio R.3. *Progettate un insieme di tabelle per una base di dati in grado di memorizzare insiemi di monete contenute in borsellini. Ciascun borsellino è caratterizzato dal nome del proprietario e da un ID identificativo univoco. Ciascun tipo di moneta ha un nome univoco e un valore. Ciascun borsellino contiene una certa quantità di monete di diversi tipi.*

Esercizio R.4. *Progettate un insieme di tabelle per una base di dati in grado di memorizzare studenti, corsi, professori e aule. Ciascuno studente segue un numero qualsiasi di corsi, eventualmente uguale a zero. Ciascun corso ha un professore, ma un professore più insegnare più corsi. Ciascun corso ha un'aula.*

Esercizio R.5. *Fornite i comandi SQL per creare una tabella Book con colonne per il codice ISBN, l'autore e il titolo, e inseritevi tutti i libri di testo che state usando.*

Esercizio R.6. *Fornite i comandi SQL per creare una tabella Car con colonne per il numero di identificazione del veicolo, il nome del costruttore, il modello e l'anno, e inseritevi tutte le automobili possedute dai componenti della vostra famiglia.*

Esercizio R.7. *Fornite un'interrogazione SQL per elencare tutti i prodotti presenti nella base di dati delle fatture.*

Esercizio R.8. *Fornite un'interrogazione SQL per elencare tutti i clienti della California.*

Esercizio R.9. *Fornite un'interrogazione SQL per elencare tutti i clienti della California o del Nevada.*

Esercizio R.10. *Fornite un'interrogazione SQL per elencare tutti i clienti che non sono delle Hawaii.*

Esercizio R.11. *Fornite un'interrogazione SQL per elencare tutti i clienti che hanno fatture non pagate.*

Esercizio R.12. *Fornite un'interrogazione SQL per elencare tutti i prodotti che sono stati acquistati da clienti della California.*

Esercizio R.13. *Fornite un'interrogazione SQL per elencare tutti gli articoli che fanno parte della fattura numero 11731.*

Esercizio R.14. *Fornite un'interrogazione SQL per calcolare la somma di tutte le quantità che fanno parte della fattura numero 11731.*

Esercizio R.15. *Fornite un'interrogazione SQL per calcolare il costo totale di tutti gli articoli della fattura numero 11731. Usate la funzione SUM(Product.Price *

Item.Quantity).

Esercizio R.16. *Fornite un enunciato di aggiornamento SQL che aumenti tutti i prezzi del 10%.*

Esercizio R.17. *Fornite un enunciato SQL che cancelli tutti i clienti della California.*

Esercizio R.18. *Scegliete un sistema di gestione di basi di dati (come DB2, Oracle, Postgres, SQL Server) e consultate la relativa documentazione sul Web:*

- *Di quale driver JDBC avete bisogno?*
- *Qual è lo URL della base di dati?*

Esercizio R.19. *Qual è la differenza tra un oggetto di tipo Connection e un oggetto di tipo Statement?*

Esercizio R.20. *Fra i comandi SQL presentati nel capitolo, quali producono un insieme di risultati, quali generano un conteggio di righe aggiornate e quali non producono nulla?*

Esercizio R.21. *In che modo un oggetto di tipo ResultSet differisce da un oggetto di tipo Iterator?*

Esercizio R.22. *Come potete rendere "a prova di bomba" il metodo deposit della classe BankAccount in modo che gli oggetti che rappresentano le connessioni e gli enunciati vengano chiusi anche quando si verifica un'eccezione? Non è semplice come potrebbe sembrare. Ad esempio, considerate il caso in cui il metodo getStatement lancia un'eccezione: in questo caso la connessione deve essere chiusa, ma il riferimento all'oggetto che rappresenta l'enunciato è ancora null.*

Esercizio R.23. *Come si può velocizzare la simulazione dello sportello bancario automatico del Paragrafo 5 in modo da evitare il costo delle frequenti connessioni alla base di dati?*

Esercizi di programmazione

Esercizio P.1. *Scrivete un programma Java che crei una tabella Coin con i nomi e i valori delle monete, inserendovi i tipi di monete seguenti: penny (un centesimo), nickel (cinque centesimi), dime (dieci centesimi), quarter (25 centesimi), half dollar (cin-quanta centesimi) e dollar (un dollaro). Il programma deve, poi, stampare la somma dei valori di tutte le monete. Usate i comandi SQL CREATE TABLE, INSERT e SELECT SUM.*

Esercizio P.2. *Scrivete un programma Java che crei una tabella Car che contenga automobili caratterizzate da: nome del costruttore, nome del modello, anno del modello ed efficienza nel consumo di carburante. Inserite alcune automobili e stampate l'efficienza media.Usate i comandi SQL CREATE TABLE, INSERT e SELECT AVG.*

Esercizio P.3. *Migliorate il programma ExecSQL in modo che le colonne visualizzate risultino allineate. Suggerimento: Usate il metodo getColumnDisplaySize della classe ResultSetMetaData.*

Esercizio P.4. *Scrivete un programma Java che usi le tabelle della base di dati per la gestione delle fatture. Chiedete all'utente il numero di una fattura e visualizzatela con un'impaginazione ragionevole.*

Esercizio P.5. *Scrivete un programma Java che usi le tabelle della base di dati per la gestione delle fatture. Generate un rapporto che elenchi tutti i clienti, tutte le fatture, le somme che sono state pagate e quelle ancora non pagate.*

Esercizio P.6. *Scrivete un programma Java che usi una base di dati di una biblioteca con i dati dei libri e dei lettori, come descritto nell'Esercizio R.2. I lettori devono essere in grado di prendere a prestito e restituire libri. Mettete a disposizione comandi per elencare i libri presi a prestito da un lettore e per trovare chi abbia preso in prestito un particolare libro. Prima di eseguire il programma, create le tabelle Patron e Book e popolatele di dati.*

Esercizio P.7. *Scrivete un programma Java che crei il registro dei voti di un corso.*

Prima di eseguire il programma, create una tabelle Student e popolatela di dati, oltre ad altre tabelle che vi possano essere utili. Il programma dovrebbe essere in grado di visualizzare tutti i voti di un certo studente e dovrebbe consentire al docente l'inserimento di un nuovo voto (come "Homework 4: 100") o la modifica di un voto già assegnato.

Esercizio P.8. *Scrivete un programma che gestisca con una base di dati un'agenda di appuntamenti. Un appuntamento è caratterizzato da: data, ora dell'inizio, ora della fine e descrizione. Ad esempio:*

Dentista 2001/10/19 17:30 18:30

Lezione Informatica 2001/10/22 08:30 10:00

Fornite un'interfaccia utente per aggiungere appuntamenti, eliminare appuntamenti annullati e visualizzare l'elenco degli appuntamenti di un certo giorno. L'interfaccia utente può essere di tipo testo o grafica.

Esercizio P.9. *Modificate il programma per la simulazione di un ATM presentato nel Paragrafo 5 in modo che visualizzi due finestre di tipo frame contenenti un ATM.*

Verificate che due utenti possano accedere contemporaneamente alla base di dati.

XML

Obiettivi del capitolo

■ *Capire elementi e attributi del linguaggio XML*

■ Capire i concetti che stanno alla base di un analizzatore sintattico (parser) XML

■ Essere in grado di leggere e scrivere documenti XML

■ Essere in grado di progettare un DTD (Document Type Definition) per documenti XML

2

XML

Le basi di dati (database) sono eccellenti per memorizzare grandi quantità di dati e per recuperarli velocemente. In questo capitolo, però, ci occupiamo di un diverso problema relativo alla memorizzazione dei dati: come trasferire dati in forma strutturata da un programma a un altro. Ovviamente, qualsiasi dato può essere trasformato in una stringa, codificandolo in qualche modo; a esempio, se volete scambiare informazioni su prodotti, potete codificare ciascun prodotto con una stringa di questo tipo: Car vacuum 24.95

Ma queste codifiche ad hoc pongono dei problemi. Ad esempio, supponete che il nome del prodotto contenga degli spazi: diventa difficile effettuare l'analisi lessicale (parsing) della stringa, cioè il destinatario dell'informazione ha qualche difficoltà a capire quale parte della stringa costituisca il nome del prodotto e quale, invece, il prezzo. In alternativa, potete usare la serializzazione di oggetti per scambiare dati binari che descrivono gli oggetti, ma anche questo approccio ha dei problemi: in particolare, il destinatario deve essere un programma Java che usi la stessa versione delle classi usata dal programma mittente.

In questo capitolo imparerete il linguaggio XML (Extensible Markup Language), un meccanismo per codificare i dati che è indipendente da qualsiasi linguaggio di programmazione. XML consente la codifica di dati complessi in una forma che il destinatario può analizzare facilmente, ed è un linguaggio che sta diventando molto diffuso per lo scambio di dati. Inoltre, è abbastanza semplice perché una grande varietà di programmi possa generare facilmente dati XML. I dati XML hanno una struttura annidata, in modo che possiate usare tale linguaggio per descrivere gerarchicamente insiemi di dati: ad esempio, una fattura che contiene molti articoli, ciascuno dei quali è

composto da un prodotto e una quantità. Poiché il formato XML è uno standard, si sono ampiamente diffuse librerie per l'analisi sintattica dei dati che, come vedrete in Il linguaggio XML consente la co-questo capitolo, sono anche facili da usare per un programmatore.

difica di dati complessi, indipen-

Leggere e scrivere documenti XML è particolarmente facile in Java, dentemente da qualsiasi linguag-al punto che, generalmente, è più facile usare il formato XML piuttosto gio di programmazione, in una

forma che il destinatario può ana-

che un formato di file ad hoc. Quindi, l'uso di XML rende i vostri pro-lizzare facilmente.

grammi più facili da scrivere e più professionali.

1

Marcatori e documenti XML

1.1

Vantaggi di XML

Il linguaggio XML si usa per codificare i dati: vediamo un esempio. Vogliamo codificare la descrizione di monete. Si potrebbe pensare di usare una codifica banale, come questa:

0.5

half dollar

Il valore e il nome della moneta si trovano su righe diverse, in modo che non si faccia confusione con gli spazi.

XML

3

Al contrario, ecco una codifica XML degli stessi dati:

```
<coin>
<value>0.5</value>
<name>half dollar</name>
</coin>
```

Il vantaggio della versione XML è evidente: potete osservare i dati e capire ciò che significano. Questo è, naturalmente, un vantaggio per il programmatore, non per il programma: un programma per computer non capisce cosa sia un "valore". Come programmatore, avete comunque il compito di scrivere il codice necessario a estrarre il valore dal contenuto

dell’elemento value, ma il fatto che il documento XML sia comprensibile a un lettore umano è un enorme vantaggio nello sviluppo di un programma.

I file XML sono leggibili sia dai

Un secondo vantaggio della versione XML consiste nella facilità di programmi per computer sia dalle

modifica: supponete che i dati della moneta cambino e che venga intro-persone.

dotto un nuovo campo, per indicare l’anno in cui la moneta è stata emessa. Nel formato banale, il campo che indica l’anno potrebbe essere aggiunto al di sotto del campo che contiene il nome: 0.5

half dollar

1982

Ora, un programma che è in grado di analizzare il vecchio formato potrebbe andare in crisi leggendo una sequenza di monete espressa nel nuovo formato: il programma potrebbe pensare che il nome sia seguito dal valore della moneta successiva. Il programma, quindi, deve essere aggiornato per poter funzionare con il nuovo formato dei dati; ora, però, il programma aggiornato non è più in grado di leggere i dati nel vecchio formato. (Ovviamente, con qualche sforzo questi problemi si possono superare, ma quando i dati diventano ancora più complessi, la gestione di diverse versioni di un formato di dati può essere difficoltosa e richiedere molto tempo.) I file di dati che usano il formato

Usando il linguaggio XML, d’altra parte, è facile aggiungere nuovi XML sono facili da modificare

elementi:

<coin>

<value>0.5</value>

<name>half dollar</name>

<year>1982</year>

</coin>

In questo caso, un programma che analizzi i nuovi dati è ancora in grado di estrarre le vecchie informazioni allo stesso modo, come contenuto degli elementi value e name. Il

programma non deve necessariamente essere aggiornato, ed è in grado di gestire diverse versioni del formato dei dati.

1.2

Differenze tra XML e HTML

Può darsi che abbiate notate che il formato XML dei dati sulle monete assomiglia, in qualche modo, al codice HTML. In particolare, le coppie di marcatori XML, come <va-4

XML

lue> e </value>, assomigliano proprio alle coppie di marcatori HMTL, come <h1> e

</h1>. Sia in XML che in HTML, i marcatori sono racchiusi da parentesi angolari, < e >, e un marcatore iniziale è accoppiato a un marcatore finale che inizia con un carattere di barra, /.

Sia HTML che XML sono deriva-

Questa somiglianza non è casuale. Sia HTML che XML sono derivati da SGML, Standard Generali-

ti da SGML (Standard Generalized Markup Language), il linguaggio zed Markup Language.

che fu sviluppato negli Anni ’70 per specificare insiemi di regole strutturali per diversi documenti e per creare documenti che rispettino tali insiemi di regole. Ad esempio, un’organizzazione che usi SGML potrebbe definire insiemi di regole per i manuali di istruzioni, i resoconti azionari, e così via. Di conseguenza, ciascun documento verrebbe predisposto secondo il suo insieme di regole, usando la sintassi SGML. Tecnicamente, il linguaggio HTML definisce uno di tali insiemi di regole: le regole strutturali per i documenti ipertestuali. Quando predispone-te un documento HTML, usate i marcatori, come <h1> o <li>, per contrassegnare (mark up) il vostro documento; i marcatori, cioè, mostrano la struttura della vostra particolare pagina Web. La struttura deve essere conforme a regole predefinite: ad esempio, un elementi li deve essere contenuto in un elenco ul o ol. Si dice che HTML è un esemplare di SGML, è un particolare insieme di regole che usa la sintassi generica di SGML.

Tuttavia, il linguaggio HTML, come viene usato oggi sul Web, non è proprio un esemplare di SGML, e ciò è forse una sfortuna. Il linguaggio SGML è molto rigido: ad esempio, se un documento

contiene un elemento li che non si trova all'interno di un elenco ul o ol, il documento non è valido e non dovrebbe essere elaborato, ma i browser visualizzano comunque l'elemento, in qualche modo. Ovviamente, ciascun produttore di browser può visualizzare un tale elemento non valido in modi diversi: trovare il modo di visualizzare l'elemento sembra "meglio" che rifiutarsi di visualizzare un documento non valido, ma in pratica è un grosso problema. Un progettista Web che controlli una pagina visualizzandola in un browser o due può pensare che la pagina sia corretta perché, per caso, tali browser la visualizzano correttamente, ma un altro browser può presentare in modo piuttosto diverso le parti non valide. Sarebbe meglio che la pagina fosse stata segnalata come non valida. Naturalmente esistono strumenti per la verifica della sintassi HTML ed è bene che li usiate per le vostre pagine Web: un esempio di tali strumenti è il verificatore di validità HTML del World Wide Web Consortium (W3C), un'organizzazione senza scopi di lucro che predispone gli standard per il Web. Potete trovare tale verificatore all'indirizzo http://

validator.w3.org/.

Il linguaggio SGML è piuttosto complesso, perché i suoi progettisti hanno voluto renderlo facile per chi digita il testo che compone i documenti. Ad esempio, quando scrivete un paragrafo che inizia con un marcatore , non è necessario che inseriate un marcatore , se la fine del paragrafo è evidente dal contesto. Se iniziate un altro paragrafo, basta che inseriate soltanto un altro marcatore e il paragrafo precedente verrà terminato automaticamente. Questo è comodo per chi digita, ma complica la logica degli insiemi di regole e l'elaborazione dei file. A causa di queste complicazioni, non è facile realizzare strumenti SGML. Il linguaggio SGML è stato molto usato in alcuni settori, come l'industria aeronautica, dove la necessità di gestire un'enorme quantità di documenti soverchia la complessità di SGML, ma non è mai diventato un linguaggio universale per la descrizione di documenti. Nel 1996, il World Wide Web Consortium iniziò a discutere sull'uso di SGML nel Web: il risultato fu XML, una XML

versione fortemente semplificata di SGML. La specifica XML 1.0 fu presentata nel 1998 ed era lunga 26 pagine, mentre la specifica di SGML è composta da più di 500

pagine. Esiste anche una versione con annotazioni che spiega bene la specifica ([1]).

Uno dei motivi della semplicità della specifica di XML sta nella sua rigidità.

■ In XML dovete fare attenzione alle lettere maiuscole e minuscole presenti nei marcatori; ad esempio, <li> e <LI> sono marcatori diversi che non hanno fra loro alcuna relazione.

■ Ciascun marcatore iniziale deve avere un marcatore finale corrispondente. Non potete omettere marcatori come </p>. Inoltre, se un marcatore non è dotato di marcatore finale, deve terminare con />, come in questo esempio

**

Quando l'analizzatore sintattico trova la sequenza />, sa che non deve cercare il marcatore finale mancante.

■ Infine, i valori degli attributi devono essere racchiusi tra virgolette. Ad esempio

* non è accettabile. Dovete usare*

* XML descrive il significato dei*

Molte persone che vedono per la prima volta XML si chiedono che aspet-dati, non come visualizzarli.

to abbia un documento XML all'interno di un browser, ma questa non è, in generale, una domanda utile da porsi. Alcuni browser fanno qualche tentativo di visualizzare documenti XML ed esiste un meccanismo di trasformazione che traduce un documento XML in formato HTML in modo che possa essere visualizzato in un browser (consultate Argomenti avanzati 2), ma la maggior parte dei dati che vengono codificati in XML non hanno niente a che vedere con i browser. Ad esempio, non sarebbe molto interessante visualizzare in un browser un documento XML che non contenga nient'altro che dati relativi a monete, come quelli visti nel paragrafo precedente. Al contrario, in questo capitolo imparerete a scrivere programmi

che analizzano dati XML. La differenza fondamentale tra XML e HTML è che XML

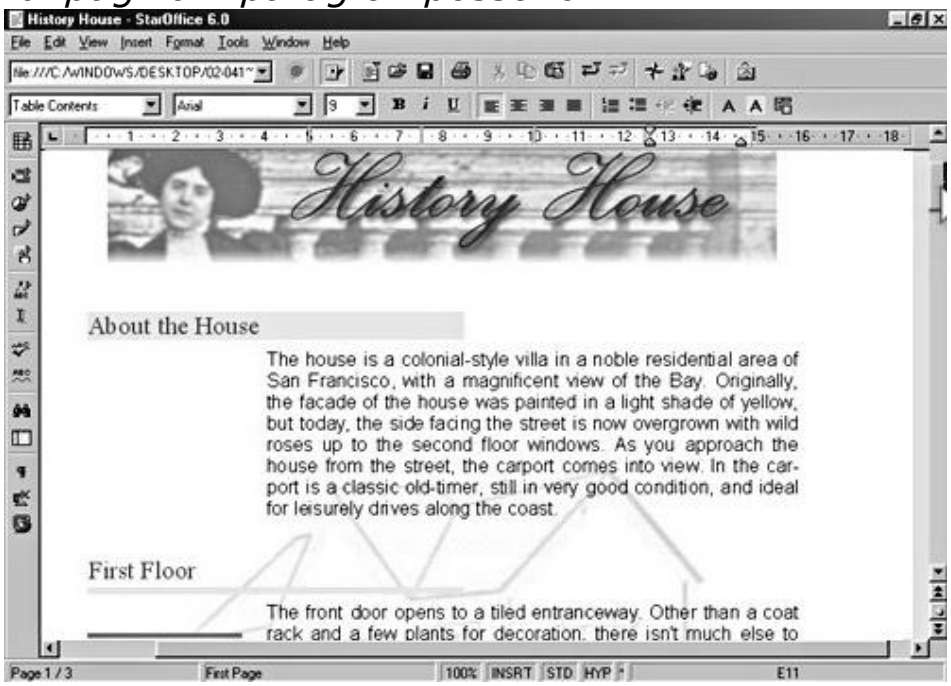
vi dice qual è il significato dei dati, non come visualizzarli.

Note di cronaca 1

Elaborazione di testi

e sistemi di videoscrittura

Quasi certamente avete utilizzato un elaboratore di testi (word processor) per scrivere lettere o resoconti. Un elaboratore di testi è un programma per scrivere e modificare documenti composti di testo e immagini. Il testo può contenere caratteri in vari font e può essere impaginato in paragrafi, tabelle e note a piè di pagina. I paragrafi possono



essere impaginati in vari modi: con il margine destro frastagliato (ragged right), cioè con i margini sinistri di tutte le righe incolonnati uno sotto l'altro, ma i margini destri non incolonnati; oppure il paragrafo può essere centrato orizzontalmente, oppure, ancora, può avere entrambi i margini completamente incolonnati (fully justified). Ciò che caratterizza gli elaboratori di testo moderni è la loro modalità operativa, detta WYSIWYG ("What You See Is What You Get", cioè "ciò che vedete è ciò che ottenete"): inserite testo e comandi, usando la

tastiera e il mouse, e lo schermo del computer visualizza immediatamente come apparirà il documento stampato (osservate la Figura 1).

Ci sono, tuttavia, degli svantaggi nella natura WYSIWYG dell'elaboratore di testo.

Può accadere che vi sforziate per sistemare varie immagini e tabelle correlate in modo che stiano insieme nella stessa pagina; più tardi, vi accorgete che dovete aggiungere un paio di paragrafi alla pagina precedente e, ora, metà del suo materiale si sposta avanti, finendo nella pagina che avevate accuratamente impaginato, costringendovi a sistemarla di nuovo. Sarebbe stato più efficiente se aveste potuto segnalare all'elaboratore di testo il vostro obiettivo, cioè: "Posiziona sempre queste immagini e tabelle insieme nella stessa pagina". In generale, i programmi WYSIWYG funzionano benis-simo nel lasciarvi sistemare il materiale come volete, ma non sanno perché avete siste-mato le cose in quel modo, per cui non sono in grado di conservare la disposizione quando il documento viene modificato. Alcuni chiamano questi programmi "What you see is all you've got", ciò che vedete è quello che siete riusciti a fare.

Ancora più importante, i programmi WYSIWYG falliscono quando avete bisogno di pubblicare lo stesso materiale in modi diversi. Può darsi che vogliate impaginare le informazioni riguardanti un prodotto sia come materiale pubblicitario sia come parte ***Figura 1***

Un elaboratore di testo
di tipo "What You See
Is What You Get"

Una sommatoria all'interno del testo: $\sum_{i=1}^{n} i^2$

La stessa sommatoria come formula evidenziata al centro:

$$\sum_{i=1}^{n} i^2$$

XML

di un elenco di componenti, oppure, ancora, può darsi che vogliate pubblicare tale informazione come modulo sul Web oppure in formato vocale per una segreteria telefonica. Non volete più "ottenere" un unico risultato, per cui non è utile vedere ciò che

"ottenete". Al contrario, diventa più importante visualizzare la struttura dell'informazione.

Un programma per scrivere e modificare testo strutturato deve memorizzare tre tipi di informazioni:

- *Il testo stesso*
- *L'elemento strutturale (paragrafo, elenco puntato, titolo, e così via) a cui appartie-ne ciascuna porzione di testo*
- *Le regole per impaginare gli elementi strutturali Per rendere agevole lo scambio di documenti strutturati fra computer diversi, l'informazione strutturale è spesso codificata con marcatori di evidenziazione (markup tags).*

Ad esempio, XML e HTML usano marcatori racchiusi tra parentesi angolari, come i ben noti marcatori <p>, <ul> e <h1>.

Negli anni '70, quando gli editori iniziarono ad abbandonare gli strumenti tradizionali di impaginazione manuale per passare a strumenti elettronici, i primi risultati furono di qualità molto inferiore, in particolar modo per le formule matematiche. Sistemare i simboli in formule complesse in un modo matematicamente sensato è un'arte che richiede pratica e obiettività, e i primi programmi di impaginazione non erano assolutamente in grado di svolgere tale compito. Frustrato da tale situazione, il famo-so scienziato informatico Donald Knuth della Stanford University decise di risolvere il problema e invento un programma di impaginazione che chiamò TEX (pronunciato

"tek" perché la lettera "X" nel nome è, in realtà, la lettera greca maiuscola chi). I dati in ingresso al programma consistono di testo con marcatori di evidenziazione che iniziano con una barra rovesciata; coppie di parentesi graffe {} vengono usate per rag-gruppare informazioni, e altri simboli speciali di

marcatura, come _ e ^, indicano pedi-ci e apici. Ad esempio, per specificare una sommatoria, scrivete

\sum_{i=1}^n i^2

Il programma TEX impagina la sommatoria come mostrato nella Figura 2. Notate che l'espressione viene impaginata in un modo quando è inserita nel testo e in un modo diverso quando è inserita in una formula visualizzata in evidenza, a sé stante.

Un marcatore di evidenziazione come <h1> in HTML o \sum in TEX è utile soprat-tutto per lo scambio di documenti tra computer con sistemi operativi diversi. Soltanto gli autori di HTML o di TEX più incalliti scrivono i marcatori a mano. In particolare, per il linguaggio HTML esistono molti programmi che visualizzano la struttura di un ***Figura 2***

Una formula

impaginata

con il sistema TEX

8

XML

documento HTML e consentono agli autori di modificare sia la struttura sia il testo in modo comodo, combinando i vantaggi dell'effetto visivo e del testo strutturato.

1.3

La struttura di un documento XML

Un insieme di dati XML viene

In questo paragrafo vedrete le regole per scrivere dati XML corretta-detto documento: inizia con un'in-

mente impaginati. Per motivi storici (cioè per le radici su cui si basa testazione (header) e contiene

l'elaborazione dei documenti in SGML), un insieme di dati XML viene elementi e testo.

detto documento, anche se la maggior parte dei dati XML ha ben poco a che fare con i documenti tradizionali.

Lo standard XML raccomanda che ogni documento XML inizi con un'intestazione:

<?xml version="1.0"?>

Quindi, il documento XML contiene i veri dati, che sono inseriti in un elemento radice.

Ad esempio:

<?xml version="1.0"?>
<purse>
altri dati
</purse>
L'elemento radice è un esempio di elemento XML. Un elemento può avere una delle due forme seguenti:
< marcatoreElemento attributiOpzionali> contenuto </ marcatoreElemento> oppure:
< marcatoreElemento attributiOpzionali/> Nel primo caso, l'elemento ha un contenuto: elementi, testo, o una combinazione di entrambi. Un valido esempio è un paragrafo di un documento HTML:
* Use XML for robust data formats. L'elemento p contiene*
1. Il testo: "Use XML for"
2. Un sottoelemento strong
3. Altro testo " data formats."
Un elemento può contenere te-
La combinazione di testo e di elementi è utile per quei file XML che sto, sottoelementi, oppure una
contengono documenti nel senso tradizionale del termine: la specifica loro combinazione ("contenuto
dello standard XML chiama contenuto misto questo tipo di contenuti. Al misto"). Nelle descrizioni di dati,
contrario, nei file che descrivono insiemi di dati (come i nostri dati sulle evitate il contenuto misto.
monete) è meglio inserire soltanto elementi che contengono testo XML
9
oppure elementi che contengono altri elementi: la specifica dello standard XML
chiama contenuto di elementi un tipo di contenuto che contiene soltanto elementi.
Ciascun elemento può contenere attributi. Ad esempio, l'elemento a del linguaggio HTML ha un attributo href che specifica lo URL di un collegamento iper-testuale:
… Gli elementi possono avere attri-Un attributo ha un nome (come href) e un valore. In

XML, il valore buti: usateli per descrivere come

deve essere racchiuso tra virgolette o singoli apici.

interpretare il contenuto degli ele-

I programmatori si chiedono spesso se sia meglio usare attributi menti.

o sottoelementi. Ad esempio, una moneta dovrebbe essere descritta così:

<coin value="0.5" name="half dollar"/>

oppure in questo altro modo:

<coin>

<value>0.5</value>

<name>half dollar</name>

</coin>

La prima forma è più sintetica, ma viola lo spirito degli attributi: gli attributi dovrebbero fornire informazioni che riguardano il contenuto degli elementi. Ad esempio, l'elemento value potrebbe avere un attributo currency che aiuti a interpretare il contenuto dell'elemento stesso. Il contenuto 0.5 ha un'interpretazione diversa nell'elemento

<value currency="USD">0.5</value>

rispetto a quella che ha nell'elemento

<value currency="EUR">0.5</value>

Un elemento può anche avere più attributi; ad esempio

* E, come avete già visto, un elemento può avere sia attributi che contenuto:*

Sun's Java Web site Fin qui, avete visto i componenti di un documento XML che servono per usare il linguaggio XML allo scopo di codificare dati. Esistono altri costrutti XML per situazioni più specifiche: consultate [1] per avere maggiori informazioni. Nel prossimo paragrafo vedrete come usare Java per effettuare l'analisi sintattica di documenti XML.

10

XML

Suggerimenti per la qualità 1

Preferite gli elementi XML

rispetto agli attributi

Gli attributi sono più brevi degli elementi. Ad esempio:
<coin value=”0.5” name=”half dollar”/>
sembra più semplice di:
<coin>
<value>0.5</value>
<name>half dollar</name>
</coin>
Si ha la tentazione di usare gli attributi perché sono “più facili da digitare”, ma, ovviamente, i documenti XML non vengono digitati, se non a scopi di collaudo. In situazioni reali, i documenti XML vengono generati da programmi.
Gli attributi sono meno flessibili degli elementi. Supponete di voler aggiungere l’indicazione della valuta al valore della moneta. Con gli elementi, ciò è facile da realizzare:
<value currency=”USD”>0.5</value>
oppure anche:
<value>
<currency>USD</currency>
<amount>0.5</amount>
</value>
Con gli attributi, siete bloccati: non potete ridefinire la struttura dei dati. Ovviamente, potete scrivere:
<coin value=”USD 0.5” name=”half dollar”/> Ma a questo punto il vostro programma deve analizzare sintatticamente la stringa USD
0.5 e scomporla manualmente, proprio il tipo di codice noioso e facile da sbagliare per evitare il quale è stato progettato il linguaggio XML.
In HTML esiste una semplice regola per decidere quando usare attributi: tutte le stringhe che non fanno parte del testo visualizzato sono attributi. Ad esempio, considerate il collegamento seguente:
The Java Web site Il testo all’interno dell’elemento a, The Java Web page, fa parte di ciò che l’utente vede sulla pagina Web, mentre la stringa http://java.sun.com dell’attributo href non viene visualizzata nella pagina.
XML

11

Ovviamente, il linguaggio HTML è un po' diverso dai documenti XML che potete costruire per descrivere dati come gli elenchi di prodotti, ma si applicano le stesse regole basilari: ogni cosa che fa parte dei dati non dovrebbe essere un attributo. Un attributo è appropriato soltanto se dice qualcosa che riguarda i dati ma non fa parte dei dati stessi. Se vi trovate coinvolti in discussioni metafisiche per decidere se una certa informazione faccia parte dei dati o spieghi qualcosa dei dati stessi, inserite tale informazione in un elemento, non in un attributo.

Suggerimenti per la qualità 2

Evitate sottoelementi con contenuto misto di elementi e di testo

I discendenti gerarchici di un elemento possono essere 1. Elementi

2. Testo

3. Una combinazione di entrambi

In HTML, mescolare elementi e testo è pratica comune, come in questo esempio:

* Use XML for robust data formats. Quando si descrivono insiemi di dati, invece, non dovreste mescolare elementi e testo. Ad esempio, non dovrete scrivere in questo modo:*

```
<value>
<currency>USD</currency>
0.5
</value>
```

Piuttosto, il contenuto di un elemento dovrebbe essere costituito da solo testo

```
<value>0.5</value>
```

oppure da soli elementi

```
<value>
<currency>USD</currency>
<amount>0.5</amount>
</value>
```

C'è una motivazione importante per questa regola di progettazione: come vedrete nel seguito di questo capitolo, è

possibile specificare regole sintattiche più rigide per gli elementi che contengono soltanto altri elementi, rispetto a ciò che si può fare con elementi che possono contenere anche testo.

12

XML

Consigli pratici 1

Progettare un formato per documenti XML

Questi Consigli Pratici descrivono passo dopo passo il procedimento di progettazione di un formato per documenti XML, mentre nel Paragrafo 4 vedrete come descrivere formalmente uno di tali formati con la definizione di un tipo di documenti. Ora, invece, concentriamo la nostra attenzione sulla definizione informale del contenuto di un documento: il "risultato" di questa attività è un documento da usare come esempio.

Passo 1. *Raccogliete gli elementi dei dati che dovete includere nel documento XML*

Scriveteli su un foglio di carta e, se è possibile, iniziate dall'analisi di alcuni esempi reali.

Ad esempio, supponete di voler progettare un documento XML per descrivere una fattura. Un fattura è caratterizzata da:

- *Un numero di fattura*
- *Un indirizzo di spedizione*
- *Un indirizzo di fatturazione*
- *Un elenco di articoli che sono stati ordinati*

Se è possibile, procurate alcune vere fatture, e decidete quali caratteristiche delle reali fatture dovete inserire nel vostro documento XML.

Passo 2. *Analizzate quali elementi dei dati hanno bisogno di essere descritti meglio Continuate a raffinare la descrizione finché non arrivate a valori dei dati che possono essere descritti da singole stringhe o numeri, e annotate tutti gli elementi di dati che avete identificato durante questo procedimento. Quando avete terminato, dovreste avere un elenco di elementi dei dati, alcuni dei quali possono essere ulteriormente scomposti, mentre altri sono abbastanza semplici da poter essere descritti da una singola stringa o da un singolo numero.*

Ad esempio, un “indirizzo di spedizione” contiene il nome del cliente e l’indicazione della via, della città, dello Stato e del codice postale (ZIP).

Un “elenco di articoli che sono stati ordinati” contiene articoli. Ciascun articolo contiene un prodotto e la quantità che ne è stata ordinata. Ciascun prodotto contiene il nome del prodotto e il suo prezzo.

Quindi, la nostra lista ora contiene:

- *Numero di fattura*
- *Indirizzo*
- *Nome del cliente*
- *Via*
- *Città*
- *Stato*

- *Codice ZIP*
- *Elenco di articoli che sono stati ordinati*
- *Articolo*
- *Prodotto*
- *Descrizione*
- *Prezzo*
- *Quantità*

Continuate a scomporre gli elementi dei dati finché ciascuno di essi può essere descritto da una sola stringa o da un solo numero. Ad esempio, un indirizzo non può essere descritto da un’unica stringa, mentre una città lo può essere.

Passo 3. *Identificate un nome di elemento adatto a caratterizzare l’intero documento XML*

Questo elemento diventa l’elemento radice. Ad esempio, i dati relativi a una fattura saranno contenuti in un elemento chiamato invoice.

Passo 4. *Identificate nomi di elementi adatti per gli elementi di primo livello che avete trovato nel Passo 1*

Questi elementi diventano i figli dell’elemento radice. Ad esempio, l’elemento invoice ha come figli

- *number*
- *address*

■ *items*

Passo 5. *Ripetete questo procedimento per assegnare un nome agli altri elementi identificati nel Passo 2*

Mentre fate ciò, costruite un esempio significativo che mostri tutti gli elementi in azione.

Nel caso della fattura, ecco un esempio di questo tipo.

```
<invoice>
<number>11365</number>
<address>
<name>John Meyers</name>
<company>ACME Computer Supplies Inc.</company>
<street>1195 W. Fairfield Rd.</street>
<city>Sunnyvale</city>
<state>CA</state>
<zip>94085</zip>
</address>
<items>
<item>
<product>
<description>Ink Jet Refill Kit</description>
```



```
<price>29.95</price>
</product>
<quantity>8</quantity>
</item>
<item>
<product>
<description>4-port Mini Hub</description>
<price>19.95</price>
</product>
<quantity>4</quantity>
</item>
</items>
</invoice>
```

Passo 6. *Controllate che il documento non abbia contenuti misti Cioè, assicuratevi che ciascun elemento abbia come figli altri elementi oppure testo, ma non entrambi. Se necessario, aggiungete ulteriori elementi come figli per incapsulare il testo.*

Ad esempio, supponete che l'elemento product fosse stato scritto così:

<product>

<description>Ink Jet Refill Kit</description> 29.95

</product>

Forse qualcuno potrebbe pensare che sia "ovvio" che l'ultimo dato rappresenti il prezzo, ma, seguendo i Consigli per la Qualità 2, è meglio incapsulare il prezzo all'interno di un elemento price, in questo modo:

<product>

<description>Ink Jet Refill Kit</description>

<price>29.95</price>

</product>

2

Analisi sintattica di documenti XML

Un parser è un programma che

Per leggere e analizzare i contenuti di un documento XML, avete biso-legge un documento, verifica che

gno di un parser (analizzatore sintattico) XML. Un parser è un program-sia sintatticamente corretto ed

ma che legge un documento, verifica che sia sintatticamente corretto ed esegue determinate azioni men-esegue determinate azioni mentre elabora il documento.

tre elabora il documento.

Vengono comunemente utilizzati due tipi di parser XML: alcuni seguono una specifica denominata SAX (Simple Access to XML), mentre altri seguono la specifica DOM (Document Object Model). Un parser SAX è guidato dagli eventi (event-driven): ogni volta che identifica un particolare costrutto sintattico (come, ad esempio, un marcatore di apertura quale <price>), invoca un metodo che dovete fornire voi. Al contrario, un parser DOM costruisce un albero che rappresenta il documento analizzato; dopo che il parser ha terminato la sua analisi, potete esami-XML

15

nare l'albero. I parser SAX hanno un vantaggio: sono più efficienti nella gestione di lunghi documenti XML, perché non devono costruire un albero. Viceversa, i parser DOM sono più

semplici da usare per la maggior parte delle applicazioni: l'albero generato dal parser vi fornisce un'immagine completa dei dati, mentre un parser SAX vi consegna l'informazione come una serie di piccole porzioni. In questo paragrafo, imparerete a usare un parser DOM.

Lo standard DOM definisce nel pacchetto org.w3c.dom un'interfac-Esistono due tipi di parser: i par-

ser SAX generano eventi mentre

cia Document che fornisce i metodi per analizzare e modificare la strut-analizzano un documento, men-

tura ad albero che rappresenta un documento XML, ma la versione at-tre i parser DOM costruiscono un

tuale dello standard DOM non specifica come trasformare un docu-albero.

mento XML in un oggetto di tipo Document. Esistono parecchi parser DOM che sono in grado di leggere un flusso d'ingresso contenente marcatori XML e trasformarlo in un oggetto di tipo Document. Nel passato, ciascun parser aveva una diversa interfaccia, rendendo difficile ai programmatori il passaggio da un parser a un altro: per risolvere questo problema, Sun Microsystems ha definito una specifica, denominata JAXP (Java API for XML Processing; ricordate, anche, che API significa Application Programming Interface). La specifica JAXP fornisce un meccanismo standard tramite il quale i parser DOM possono leggere e creare documenti, e fa parte della versione Java 1.4 e successive. Se usate una versione precedente di Java, dovete scaricare librerie addizionali.

Un oggetto di tipo Document-

Come già detto, l'interfaccia Document descrive la struttura ad albero Builder è in grado di leggere

di un documento XML. Per generare un esemplare di una classe che un documento XML da un file,

implementi l'interfaccia Document,vi serve un oggetto di tipo Document-da uno URL o da un flusso di in-

Builder, che potete costruire invocando il metodo statico newInstance gresso. Il risultato è un oggetto

di tipo Document, che contiene

della classe DocumentBuilderFactory e, successivamente, invocando il un albero.

metodo newDocumentBuilder con l'oggetto di tipo DocumentBuilderFactory.

DocumentBuilderFactory factory =

DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();
Dopo aver costruito un oggetto di tipo DocumentBuilder, potete leggere un documento. Per leggere un documento da un file, per prima cosa costruite un oggetto di tipo File a partire da un nome di file, poi invocate il metodo parse della classe DocumentBuilder.

String fileName = …;

File f = new File(fileName);

Document doc = builder.parse(f);

Se il documento si trova sulla rete Internet, usate uno URL invece di un file: String urlName = …;

URL u = new URL(urlName);

Document doc = builder.parse(u);

Oppure, potete anche leggere un documento da un flusso di ingresso qualsiasi: InputStream in = …;

Document doc = builder.parse(in);

16

XML

Dopo aver creato un nuovo documento o aver letto un documento da un file, potete esaminarlo e modificarlo, iniziando dall'elemento radice, che viene restituito dal metodo getDocumentElement dell'interfaccia Document.

Element root = doc.getDocumentElement();

L'albero che rappresenta un do-

Come Document, anche il tipo Element è un'interfaccia, non una classe: cumento è composto di nodi. I

il costruttore di documenti ha il compito di produrre un oggetto che tipi di nodi più importanti sono

implementi tale interfaccia. La stessa considerazione vale per tutti i tipi Element e Text.

definiti dallo standard DOM: sono tutti interfacce.

L’interfaccia Element è un’interfaccia derivata dall’interfaccia più generica Node e, oltre all’interfaccia Element, esistono parecchi altri tipi di nodi che sono interfacce derivate dall’interfaccia Node. In questo capitolo, l’unica altra interfaccia derivata che ci interessa è l’interfaccia Text (osservata la Figura 3).

Se un elemento ha soltanto uno o due figli, potete usare i metodi getFirstChild e getLastChild per recuperare un solo figlio; tali metodi restituiscono null se il nodo richiesto non esiste.

Per visitare i nodi figli di un ele-

Tuttavia, la maggior parte degli elementi ha più di uno o due figli, mento, usate un oggetto di tipo

per cui usate il metodo getChildNodes dell’interfaccia Element per ot-NodeList.

tenere gli elementi figli di un elemento, memorizzati in un esemplare di una classe che implementa l’interfaccia NodeList. Ad esempio, considerate l’elenco di articoli della Figura 4: considerate tale documento come se fosse un albero la cui radice è un elemento di tipo items (osservate la Figura 5). Applicando il metodo getChildNodes all’elemento radice, si può ottenere un elenco di nodi che contiene due elementi di tipo item.

Figura 3

Diagramma UML
delle interfacce
dello standard DOM
usate in questo
capitolo


```
<?xml version="1.0"?>
<items>
<item>
<product>
<description>Ink Jet Refill Kit</description>
<price>29.95</price>
</product>
<quantity>8</quantity>
```

</item>
<item>
<product>
<description>4-port Mini Hub</description>
<price>19.95</price>
</product>
<quantity>4</quantity>
</item>
Figura 4
</items>
Un documento XML
Figura 5
Vista ad albero
del documento

Purtroppo, un oggetto di tipo NodeList non implementa l'interfaccia standard List del pacchetto java.util, per cui, per usarlo, dovete imparare un altro insieme di metodi. Usate il metodo getLength per conoscere il numero di nodi nella lista, poi chiamate il metodo item per recuperare un elemento dalla lista di nodi. Ecco come si accede all'elemento figlio della radice con indice i.

```
NodeList nodes = root.getChildNodes();
int i = ...; // un valore compreso tra 0 e nodes.length() - 1
Node childNode = nodes.item(i);
```


Per impostazione predefinita, il parser XML conserva tutte le spaziature (cioè: spazi, caratteri di tabulazione e caratteri di andata a capo) presenti fra elementi, nel caso li vogliate analizzare. Di conseguenza, i figli di un elemento comprendono elementi e testo (le spaziature). Quando il vostro documento XML descrive dati, non vi importa delle spaziature; se avete avuto cura di non combinare elementi con testo avente significato, come descritto in Consigli per la Qualità 2, ignorare le spaziature è facile: se state elaborando un elemento i cui figli sono a loro volta elementi (come l'elemento item nel nostro esempio), semplicemente ignorate tutti i nodi figli che non siano nodi di tipo elemento. Usate il ciclo seguente:

for (int i = 0; i < nodes.getLength(); i++)
{
Node childNode = nodes.item(i);
if (childNode instanceof Element)
{
Element childElement = (Element)childNode;
// elabora childElement
}
}
Se non usate un DTD, il parser
Nel Paragrafo 5, vedrete come indicare al parser di non considerare le XML conserva tutte le spazia-spaziature: per farlo, dovrete fornire una definizione di un tipo di docu-ture.
mento (Document Type Definition, DTD) che specifichi la struttura del vostro documento XML. Per ora, è più semplice inserire una verifica per ignorare le spaziature.
Dopo aver recuperato un elemento, potete ottenerne il nome (come price) invocando il metodo getTagName.
Element priceElement = ...;
String name = priceElement.getTagName();
// restituisce il nome di un marcatore, come price Potete anche recuperare gli attributi per nome. Ad esempio, ecco come cercare l'attributo currency:
String attributeValue = priceElement.getAttribute("currency"); Per scandire gli attributi di un ele-Se non sapete il nome di un attributo, potete iterare su una lista di tutti mento, usate un oggetto di tipo
gli attributi, usando un oggetto di tipo NamedNodeMap che, come l'inter-NamedNodeMap. Ciascun attributo
faccia NodeList, è un'interfaccia peculiare della libreria DOM. L'inter-
è memorizzato in un oggetto di
faccia NamedNodeMap è concettualmente simile all'interfaccia standard tipo Node.
Map di Java, ma i dettagli sulle modalità di utilizzo sono diversi. Il metodo getAttributes restituisce una mappa di tutte le coppie di nomi e valori degli attributi:

NamedNodeMap attributes = priceElement.getAttributes(); Il metodo getLength vi dice quanti attributi sono presenti nell'elemento; per recuperare un particolare attributo, invocate il metodo item, proprio come fareste con un XML

19

oggetto di tipo NodeList. Quindi, chiamate i metodi getNodeName e getNodeValue, applicandoli all'oggetto di tipo Node restituito: int i = ...; // un valore compreso tra 0 e attributes.getLength() - 1

Node anAttributeNode = attributes.item(i);

String attributeName = anAttributeNode.getNodeName();

// esempio: "currency"

String attributeValue = anAttributeNode.getNodeValue();

// esempio: "USD"

Ora sapete come analizzare elementi e i loro attributi, e come trovare i loro figli se sono essi stessi elementi. Tuttavia, alcuni elementi hanno figli che contengono testo: nel nostro esempio, si tratta degli elementi description, price e quantity, e il costruttore di documenti crea nodi di tipo Text per tali figli. Se seguite i Consigli per la Qualità 2, tutti gli elementi che contengono testo hanno un unico nodo figlio di tipo Text, che potete recuperare con il metodo getFirstChild. Per leggere il testo contenuto in tale nodo, usate il metodo getData, che restituisce una stringa, eventualmente da convertire in un numero. Ad esempio, ecco come determinare il prezzo memorizzato in un elemento price:

Element priceNode = ...;

Text priceData = (Text)priceNode.getFirstChild(); String priceString = priceData.getData(); // esempio: "24.95"

double price = Double.parseDouble(priceString);

Ora sapete come leggere e analizzare un documento XML. Il programma d'esempio al termine di questo paragrafo mette all'opera queste tecniche: la classe ItemListParser è in grado di analizzare un documento XML che contiene un elenco di descrizioni di prodotti, e il suo metodo parse riceve un nome di file e restituisce un vettore di oggetti di tipo Item.

ItemListParser parser = new ItemListParser();

ArrayList items = parser.parse("items.xml");

La classe ItemListParser traduce ciascun elemento di tipo item in un esemplare della classe Java Item e ciascun elemento di tipo product in un esemplare della classe Java Product. L'elemento di tipo items viene trasformato in un ArrayList di articoli. A questo scopo, si usano dei metodi statici ausiliari:

```
private Product getProduct(Element e)
private Item getItem(Element e)
private ArrayList getItems(Element e)
```

Tuttavia, non ci sono metodi ausiliari per analizzare gli elementi description, price e quantity: l'analisi di questi elementi è così semplice che viene svolta all'interno dei metodi getProduct e getItem. In generale, è bene creare un metodo ausiliario per ciascun elemento XML che contiene sottoelementi.

File ItemListParser.java

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory; import javax.xml.parsers.ParserConfigurationException; import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Text;
import org.xml.sax.SAXException;
/**
Un parser XML per elenchi di articoli.
*/
public class ItemListParser
{
/**
Costruisce un parser in grado di analizzare elenchi di articoli.
```

```
*/
public ItemListParser()
throws ParserConfigurationException
{
DocumentBuilderFactory factory
= DocumentBuilderFactory.newInstance();
builder = factory.newDocumentBuilder();
}
/**
Analizza un file XML contenente un elenco di articoli.
@param fileName il nome del file
@return un vettore contenente tutti gli articoli del file XML
*/
public ArrayList parse(String fileName)
throws SAXException, IOException
{
File f = new File(fileName);
Document doc = builder.parse(f);
// recupera l'elemento radice <items>
Element root = doc.getDocumentElement();
return getItems(root);
}
/**
Costruisce un vettore di articoli estratti da un elemento DOM.
@param e un elemento di tipo <items>
@return un vettore di tutti i figli di tipo <item> di e
*/
```



```
private static ArrayList getItems(Element e)
{
ArrayList items = new ArrayList();
// recupera i figli di tipo <item>
NodeList children = e.getChildNodes();
for (int i = 0; i < children.getLength(); i++)
{
Node childNode = children.item(i);
```

```
if (childNode instanceof Element)
{
Element childElement = (Element)childNode;
if (childElement.getTagName().equals("item"))
{
Item c = getItem(childElement);
items.add(c);
}
}
}
return items;
}
/**
Costruisce un articolo estratto da un elemento DOM.
@param e un elemento di tipo <item>
@return l'articolo descritto dall'elemento
*/
private static Item getItem(Element e)
{
NodeList children = e.getChildNodes();
Product p = null;
int quantity = 0;
for (int j = 0; j < children.getLength(); j++)
{
Node childNode = children.item(j);
if (childNode instanceof Element)
{
Element childElement = (Element)childNode;
String tagName = childElement.getTagName();
if (tagName.equals("product"))
p = getProduct(childElement);
else if (tagName.equals("quantity"))
{
Text textNode =
= (Text)childElement.getFirstChild();
String data = textNode.getData();
quantity = Integer.parseInt(data);
}
```

```
}
}
return new Item(p, quantity);
}
```

22

XML

```
/**
Costruisce un prodotto estratto da un elemento DOM.
@param e un elemento di tipo <product>
@return il prodotto descritto dall'elemento
*/
private static Product getProduct(Element e)
{
NodeList children = e.getChildNodes();
String name = "";
double price = 0;
for (int j = 0; j < children.getLength(); j++)
{
Node childNode = children.item(j);
if (childNode instanceof Element)
{
Element childElement = (Element)childNode;
String tagName = childElement.getTagName();
Text textNode
= (Text)childElement.getFirstChild();
String data = textNode.getData();
if (tagName.equals("description"))
name = data;
else if (tagName.equals("price"))
price = Double.parseDouble(data);
}
}
return new Product(name, price);
}
private DocumentBuilder builder;
}
```

File Item.java

```
/**
```

Descrive la quantità di un articolo acquistato e il suo prezzo.
*/
class Item
{
*/**
Costruisce un articolo con un prodotto e una quantità.
@param aProduct il prodotto
@param aQuantity la quantità dell'articolo
*/
public Item(Product aProduct, int aQuantity)
{
theProduct = aProduct;
quantity = aQuantity;
}
*/**
Calcola il prezzo totale di questo articolo.
@return il prezzo totale
*/
XML
23
public double getTotalPrice()
{
*return theProduct.getPrice() * quantity;*
}
*/**
Impagina l'articolo.
@return l'articolo impaginato
*/
public String format()
{
final int COLUMN_WIDTH = 30;
String description = theProduct.getDescription(); String r = description;
// inserisci spazi per riempire la colonna
int pad = COLUMN_WIDTH – description.length();
for (int i = 1; i <= pad; i++)
r = r + " ";
r = r + theProduct.getPrice()

```
+ “ “ + quantity
+ “ “ + getTotalPrice();
return r;
}
private int quantity;
private Product theProduct;
}
```

File Product.java

```
/**
Descrive un prodotto avente una descrizione e un prezzo.
*/
class Product
{
/**
Costruisce un prodotto con una descrizione e un prezzo.
@param aDescription la descrizione del prodotto
@param aPrice il prezzo del prodotto
*/
public Product(String aDescription, double aPrice)
{
description = aDescription;
price = aPrice;
}
/**
Fornisce la descrizione del prodotto.
@return la descrizione
*/
```



```
public String getDescription()
{
return description;
}
/**
Fornisce il prezzo del prodotto.
@return il prezzo unitario
*/
public double getPrice()
```

```
{
return price;
}
private String description;
private double price;
}
```

File ItemListParserTest.java

```
import java.util.ArrayList;
/**
Questo programma analizza un file XML contenente un elenco di articoli
e visualizza gli articoli descritti nel file.
*/
public class ItemListParserTest
{
public static void main(String[] args)
throws Exception
{
ItemListParser parser = new ItemListParser();
ArrayList items = parser.parse("items.xml");
for (int i = 0; i < items.size(); i++)
{
Item anItem = (Item)items.get(i);
System.out.println(anItem.format());
}
}
}
```

Consigli pratici 2

Scrivere un parser per un tipo di documento XML

Se vi viene richiesto di scrivere un parser, dovete scrivere una classe che possa essere utilizzata per leggere un documento XML e produrre un oggetto Java di un tipo specifico:

XML


```
public class MyParser
{
public SomeClass parse(String fileName)
```

```
...
}
```

Passo 1. *Osservate il formato del documento XML*

Considerate tutti gli elementi aventi altri elementi come contenuto (cioè, elementi che contengono sottoelementi) e, per ciascun di essi, chiedetevi a quale classe dovrebbe appartenere: potrebbe essere una classe Java già esistente, oppure una classe che dovete scrivere voi, oppure la classe ArrayList, nel caso in cui l'elemento contenga semplicemente una sequenza di elementi figli che siano tutti dello stesso tipo.

Considerate l'esempio della fattura dei Consigli pratici 1. Gli elementi che hanno altri elementi come contenuto sono i seguenti:

- *invoice*
- *address*
- *items*
- *item*
- *product*

I marcatori invoice, address, item e product corrispondono ai tipi Java Invoice, Address, Item e Product, mentre il marcatore items contiene una sequenza di oggetti di tipo item, per cui possiamo usare, semplicemente, un oggetto di tipo ArrayList.

Passo 2. *Per ogni classe identificata al Passo 1, aggiungete alla classe del vostro parser un metodo statico di tipo get*

Il metodo get dovrebbe avere una struttura simile a questa:

```
private static ClasseElemento getNomeElemento(Element e)
{
NodeList children = e.getChildNodes();
for (int i = 0; i < children.getLength(); i++)
{
Node childNode = children.item(i);
if (childNode instanceof Element)
{
Element childElement = (Element)childNode;
converti l'elemento figlio in un oggetto Java
}
}
```

usa i valori degli elementi figli per costruire un oggetto di tipo ClasseElemento e restituiscilo

}

Esaminando gli elementi figli, ci sono due casi distinti. Se l'elemento figlio contiene testo, allora dovete convertire i dati contenuti nell'elemento: Text textNode = (Text)childElement.getFirstChild(); String data = textNode.getData();

Se necessario, convertite tale stringa in un numero, usando Integer.parseInt oppure Double.parseDouble.

Se l'elemento figlio non contiene testo, allora invocate il suo metodo getNomeElementoFiglio:

ClasseElementoFiglio childObject =

getNomeElementoFiglio(childElement);

Ad esempio, ecco il metodo getItem completo, relativo all'elemento di tipo item: private static Item getItem(Element e)

{

Product p = null;

int quantity = 0;

NodeList children = e.getChildNodes();

for (int i = 0; i < children.getLength(); i++)

{

Node childNode = children.item(i);

if (childNode instanceof Element)

{

Element childElement = (Element)childNode;

// converti l'elemento figlio in un oggetto Java if (childElement.getTagName().equals("quantity"))

{

// recupera i dati del testo e convertili in un numero Text textNode

= (Text)childElement.getFirstChild();

String data = textNode.getData();

quantity = Integer.parseInt(data);

}

else if (childElement.getTagName().equals("product"))

{
// recupera l'oggetto contenuto nell'elemento figlio p = getProduct(childElement);
}
}
}
// usa i valori degli elementi figli per costruire un oggetto
// di tipo ClasseElemento e restituiscilo
Item r = new Item(quantity, p);
return r;
}
Quindi, la classe InvoiceParser ha cinque metodi ausiliari: public class InvoiceParser
{
...
private static Invoice getInvoice(Element e) {...}
private static Address getAddress(Element e) {...}
private static ArrayList getItems(Element e) {...}
private static Item getItem(Element e) {...}
private static Product getProduct(Element e) {...}
}
XML
27
Vi può essere utile realizzare questi metodi con uno stile "bottom up", partendo cioè dal metodo più semplice (come getProduct) e terminando con il metodo per l'elemento radice (getInvoice).
Passo 3. *Completate il parser scrivendo un costruttore e il metodo parse public class MyParser*
{
public MyParser()
throws ParserConfigurationException
{
DocumentBuilderFactory factory
= DocumentBuilderFactory.newInstance();
builder = factory.newDocumentBuilder();
}

public ClasseElementoRadice parse(String fileName) throws SAXException, IOException

{

File f = new File(fileName);

Document doc = builder.parse(f);

Element root = doc.getDocumentElement();

return getNomeElementoRadice(root);

}

// metodi ausiliari

...

private DocumentBuilder builder;

}

Ad esempio, nell'ultima riga del metodo parse della classe InvoiceParser, scrivete: return getInvoice(root);

Errori comuni 1

Gli elementi XML descrivono variabili istanza, non classi

I Consigli Pratici 2 spiegano come convertire documenti XML in classi Java: dovete identificare una classe per ciascun tipo di elemento. Ciò può creare confusione; ad esempio, considerate una descrizione leggermente diversa di una fattura, con indirizzi di spedizione e fatturazione diversi:

<invoice>

<number>11365</number>

<shipto>

<name>John Meyers</name>

<company>ACME Computer Supplies Inc.</company>

<street>1195 W. Fairfield Rd.</street>

<city>Sunnyvale</city>

<state>CA</state>

<zip>94085</zip>

</shipto>

<billto>

<name>Accounts Payable</name>

<company>ACME Computer Supplies Inc.</company>

<street>P.O. Box 11098</street>

<city>Sunnyvale</city>
<state>CA</state>
<zip>94080-1098</zip>
</billto>
<items>
...
</items>
</invoice>

Dovreste avere una classe Shipto corrispondente all'elemento shipto e un'altra classe Billto corrispondente all'elemento billto? Ciò non ha senso, perché hanno entrambe lo stesso contenuto: elementi che descrivono un indirizzo.

Dovete, invece, pensare agli elementi XML come all'equivalente di variabili istanza e, di conseguenza, identificare una classe appropriata. Ad esempio, un oggetto di tipo fattura ha le seguenti variabili istanza:

- *billto, di tipo Address*
- *shipto, di tipo Address*

Notate che nel documento XML non vedete le classi: nel documento XML che descrive la fattura non esiste niente che si riferisca alla classe Address. Per rendere esplicite le classi degli elementi, si usa uno schema XML, sul quale potete trovare maggiori informazioni in Argomenti avanzati 1.

Note di cronaca 2

Grammatiche, parser e compilatori

Le grammatiche sono molto importanti in molte aree dell'informatica, per descrivere la struttura di programmi per calcolatore o il formato dei dati. Per presentare il concetto di grammatica, consideriamo questo insieme di regole per un insieme di semplici frasi in linguaggio inglese:

1. Una frase ha una parte nominale seguita da un verbo e da un'altra parte nominale.

2. Una parte nominale è composta da un articolo seguito da un elenco di aggettivi, seguiti da un sostantivo.

3. Un elenco di aggettivi è costituito da un aggettivo oppure da un aggettivo seguito da un elenco di aggettivi.

4. Gli articoli sono “a” e “the”.
5. Gli aggettivi sono “quick”, “brown”, “lazy” e “hungry”.
6. I sostantivi sono “fox”, “dog” e “hamster”.
7. I verbi sono “jumps over” e “eats”.

Ecco due frasi che rispettano tali regole:

- *The quick brown fox jumps over the lazy dog.*
- *The hungry hamster eats a quick brown fox.*

Usando dei simboli, queste regole si possono esprimere con una grammatica formale:

<sentence> ::= <noun-phrase> <verb> <noun-phrase>
<noun-phrase> ::= <article> <adjective-list> <noun>
<adjective-list> ::=
<adjective> | <adjective> <adjective-list>
<article> ::= a | the
<adjective> ::= quick | brown | lazy | hungry
<noun> ::= fox | dog | hamster
<verb> ::= jumps over | eats

Qui, il simbolo ::= significa “può essere sostituito con”. Ad esempio, <article> può essere sostituito con a oppure con the.

I simboli della grammatica, come <noun>, sono racchiusi tra parentesi angolari come i marcatori XML, ma sono diversi. Uno degli scopi di una grammatica è la generazione di stringhe che siano valide secondo la grammatica stessa, iniziando con il simbolo iniziale (<sentence>, in questo esempio) e applicando regole di sostituzione finché la stringa risultante sia priva di simboli. Ad esempio: ***Stringa***

Regola

<sentence>
Inizio
<noun-phrase> <verb> <noun-phrase> 1
<noun-phrase> eats <noun-phrase>
7
<article> <adjective-list> <noun> eats <noun-phrase> 2
the <adjective-list> <noun> eats <noun-phrase> 4
the <adjective> <noun> eats <noun-phrase> 3
the hungry <noun> eats <noun-phrase> 5
the hungry hamster eats <noun-phrase>
6

the hungry hamster eats <article> <adjective-list> <noun> 2

the hungry hamster eats a <adjective-list> <noun> 4

the hungry hamster eats a <adjective> <adjective-list> <noun> 3

the hungry hamster eats a quick <adjective-list> <noun> 5

the hungry hamster eats a <adjective> <noun> 3

the hungry hamster eats a quick brown <noun> 5

the hungry hamster eats a quick brown fox

6

30

XML

Figura 6

Albero sintattico

per una semplice frase

Se avete una grammatica e una stringa, come “the hungry hamster eats a quick brown fox” oppure “a brown jumps over hamster quick lazy”, potete analizzare sintatticamente (parse) la frase, cioè verificare se la frase sia descritta dalle regole della grammatica e, se lo è, mostrare come la si può derivare a partire dal simbolo iniziale.

Un modo per mostrare tale derivazione è la costruzione di un albero sintattico (parse tree), come potete osservare nella Figura 6.

Un parser (analizzatore sintattico) è un programma che legge stringhe e decide se sono conformi alle regole di una determinata grammatica. Alcuni parser, come il parser XML DOM, costruiscono un albero sintattico durante la loro elaborazione e se-gnalano un errore quando non è possibile costruirlo. Altri parser, come il parser XML

SAX, invocano metodi specificati dall’utente ogni volta che una porzione della stringa d’ingresso è stata analizzata con successo.

L’uso più importante dei parser si trova nei compilatori per i linguaggi di programmazione. Così come la nostra grammatica può descrivere (alcune) semplici frasi in lingua inglese, le “frasi” lecite in un linguaggio di programmazione possono essere descritte da una grammatica. La grammatica del linguaggio di

programmazione Java occupa 15 pagine nella Java Language Specification [2]; per dare un'idea di un piccolo sottoinsieme di tale grammatica, ecco la descrizione delle espressioni aritmetiche.

<expression> ::=

<term> | <term> <additive-operator> <expression>

<additive-operator> ::= + | –

<term> ::=

Figura 7

Albero sintattico

per un'espressione

<factor> | <factor> <multiplicative-operator> <term>

*<multiplicative-operator> ::= * | /*

<factor> ::= <integer> | (<expression>)

<integer> ::= <digits> | – <digits>

<digits> ::= <digit> | <digit> <digits>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Un esempio di espressione lecita in questa grammatica è:

*–2 * (3 + 10)*

La Figura 7 mostra l'albero sintattico per questa espressione.

In un compilatore, l'analisi sintattica del programma sorgente è il primo passo verso la generazione del codice che possa essere eseguito da un particolare processore (la macchina virtuale Java nel caso del linguaggio Java). Scrivere un parser è un compito interessante e complesso; può darsi che, a un certo punto dei vostri studi, seguia-te un corso di progettazione di compilatori, nel quale imparerete come scrivere un parser e come generare codice a partire da ciò che viene riconosciuto dal parser. Fortunatamente, per usare il linguaggio XML non avete bisogno di sapere come il parser svolga il suo lavoro: semplicemente, chiedete al parser XML di leggere un documento XML in ingresso ed esaminate l'albero Document risultante.

Creare documenti XML

Nel paragrafo precedente avete visto come leggere un documento XML, generando un oggetto di tipo Document, e come analizzare i contenuti di tale oggetto. In questo paragrafo vedrete come fare il contrario: costruire un oggetto di tipo Document e sal-varlo in un documento XML. Ovviamente, potete creare un documento XML usando semplicemente una sequenza di enunciati print, ma non è una buona idea: in questo modo, è facile costruire un documento XML non valido, magari perché i nomi contengono caratteri speciali come < oppure &.

Ricordate che per leggere un documento XML avevate bisogno di un oggetto di tipo DocumentBuilder: ne avete bisogno anche per creare un nuovo documento, vuoto. Quindi, per creare un nuovo documento, dapprima create una fabbrica di costrut-tori di documenti, poi un costruttore di documenti, infine il documento vuoto: DocumentBuilderFactory factory =

DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder(); Document doc = builder.newDocument();

// un documento vuoto

L'interfaccia

Ora siete pronti per inserire nodi nel documento. L'oggetto di tipo do-Document ha me-

todi per creare elementi e nodi

cumento è una fabbrica di nodi: usare i metodi createElement e crea-di testo.

teTextNode dell'interfaccia Document per creare gli elementi e i nodi di testo di cui avete bisogno. Al metodo createElement dovete fornire il nome di un marcatore:

XML

33

Element itemElement = doc.createElement("item"); Al metodo createTextNode, invece, fornite una stringa di testo: Text quantityText = doc.createTextNode("8");

Gli attributi di un elemento si impostano con il metodo setAttribute. Ad esempio: priceElement.setAttribute("currency", "USD");

Per costruire la struttura ad albero di un documento, iniziate dalla radice e aggiunge-tevi figli con il metodo appendChild. Ad esempio, qui di seguito costruiamo un albero XML che descrive un articolo (osservate la Figura 8).

```
// crea gli elementi
Element itemElement = doc.createElement("item"); Element productElement = doc.createElement("product"); Element descriptionElement
= doc.createElement("description");
Element priceElement = doc.createElement("price"); Element quantityElement
= doc.createElement("quantity");
Text descriptionText
= doc.createTextNode("Ink Jet Refill Kit");
Text priceText = doc.createTextNode("29.95");
Text quantityText = doc.createTextNode("8");
// aggiungi gli elementi al documento
doc.appendChild(itemElement);
itemElement.appendChild(productElement);
itemElement.appendChild(quantityElement);
productElement.appendChild(descriptionElement);
productElement.appendChild(priceElement);
descriptionElement.appendChild(descriptionText);
priceElement.appendChild(priceText);
quantityElement.appendChild(quantityText);
```

Figura 8:
Albero XML che descrive un articolo

XML

Nel momento in cui viene scritto questo libro, lo standard DOM non prevede nessun metodo "ufficiale" per scrivere in un flusso un oggetto di tipo Document: una mancanza sorprendente, perché generare un documento XML è un'azione molto frequente.

(La versione 3 dello standard, attualmente in fase di sviluppo, dovrebbe prevedere un metodo per questo scopo.)

Nel frattempo, il modo più semplice per scrivere un documento XML è l'utilizzo di un paio di classi presenti nella libreria di trasformazioni XML (sulla quale potete avere maggiori informazioni consultando Argomenti Avanzati 2). Fortunatamente, non c'è bisogno di sapere nulla sulle trasformazioni XML per farlo; create semplicemente un oggetto di tipo Transformer, in questo modo:

Transformer t =

TransformerFactory.newInstance().newTransformer(); Per scrivere un documento XML

Successivamente, trasformate il vostro oggetto di tipo Document in un in un flusso usate un oggetto di

oggetto di tipo DOMSource e il vostro flusso di uscita in un oggetto di tipo tipo Transformer.

StreamResult, invocando poi il metodo transform: t.transform(new DOMSource(doc),

new StreamResult(System.out));

L'oggetto di tipo Transformer scrive il documento XML senza spazi e senza mai andare a capo, quindi il risultato che si ottiene sembra peggiore di quanto si sarebbe scritto manualmente, ma in realtà è più adatto all'analisi sintattica da parte di un altro programma, proprio perché non contiene spaziature non necessarie.

Il programma di esempio al termine di questo paragrafo mostra come costruire e visualizzare un documento XML. Il programma usa una classe ItemListBuilder che trasforma un vettore di articoli in un documento XML.

La classe ItemListBuilder ha alcuni metodi ausiliari: private Element createItemList(ArrayList items)

private Element createItem(Item anItem)

private Element createProduct(Product p)

Notate che questi metodi svolgono la funzione inversa dei metodi ausiliari della classe ItemListParser vista nel paragrafo precedente. I metodi ausiliari della classe del parser trasformano elementi in oggetti Java, mentre i metodi ausiliari del costruttore di documenti trasformano oggetti Java in elementi. Come con la classe ItemListParser, usiamo un metodo ausiliario per ogni elemento XML che ha elementi come figli.

File ItemListBuilder.java

import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory; import javax.xml.parsers.ParserConfigurationException; import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Text;

XML

35

*/**

Costruisce un documento DOM a partire da un vettore di articoli.

/

public class ItemListBuilder

{

*/**

Costruisce un costruttore di elenchi di articoli.

/

public ItemListBuilder()

throws ParserConfigurationException

{

DocumentBuilderFactory factory

= DocumentBuilderFactory.newInstance();

builder = factory.newDocumentBuilder();

}

*/**

Costruisce un documento DOM a partire da un elenco di articoli.

@param items gli articoli

@return un documento DOM che descrive gli articoli

/

public Document build(ArrayList items)

{

doc = builder.newDocument();

Element root = createItemList(items);

doc.appendChild(root);

return doc;

}
*/**
Costruisce un elemento DOM a partire da un elenco di articoli.*
@param items gli articoli
@return un elemento DOM che descrive gli articoli
/
private Element createItemList(ArrayList items)
{
Element itemsElement = doc.createElement("items"); for (int i = 0; i < items.size(); i++)
{
Item anItem = (Item)items.get(i);
Element itemElement = createItem(anItem);
itemsElement.appendChild(itemElement);
}
return itemsElement;
}
*/**
Costruisce un elemento DOM che rappresenta un articolo.*
@param anItem l'articolo
@return un elemento DOM che descrive l'articolo
/
private Element createItem(Item anItem)
{
Element itemElement = doc.createElement("item"); ***36***
XML
Element productElement
= createProduct(anItem.getProduct());
Text quantityText = doc.createTextNode(
"" + anItem.getQuantity());
Element quantityElement
= doc.createElement("quantity");
quantityElement.appendChild(quantityText);
itemElement.appendChild(productElement);
itemElement.appendChild(quantityElement);
return itemElement;
}

```
/**
Costruisce un elemento DOM che rappresenta un prodotto.
@param p il prodotto
@return un elemento DOM che descrive il prodotto
*/
private Element createProduct(Product p)
{
Text descriptionText
= doc.createTextNode(p.getDescription());
Text priceText
= doc.createTextNode("" + p.getPrice());
Element descriptionElement
= doc.createElement("description");
Element priceElement
= doc.createElement("price");
descriptionElement.appendChild(descriptionText);
priceElement.appendChild(priceText);
Element productElement
= doc.createElement("product");
productElement.appendChild(descriptionElement);
productElement.appendChild(priceElement);
return productElement;
}
private DocumentBuilder builder;
private Document doc;
}
```

File ItemListBuilderTest.java

```
import java.util.ArrayList;
import org.w3c.dom.Document;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
```



```
/**
Questo programma collauda il costruttore di elenchi di articoli.
```

Visualizza in uscita il file XML corrispondente a un documento DOM che contiene un elenco di articoli.

```
*/
public class ItemListBuilderTest
{
public static void main(String[] args)
throws Exception
{
ArrayList items = new ArrayList();
items.add(new Item(
new Product("Toaster", 29.95), 3));
items.add(new Item(
new Product("Hair dryer", 24.95), 1));
ItemListBuilder builder = new ItemListBuilder(); Document doc = builder.build(items);
Transformer t =
TransformerFactory.newInstance().new Transformer(); t.transform(new DOMSource(doc),
new StreamResult(System.out));
}
}
```

Questo programma usa le classi Item e Product del paragrafo precedente. La classe Item è stata modificata aggiungendo i metodi getProduct e getQuantity.

Consigli pratici 3

Scrivere un documento XML

Ovviamente, potreste scrivere un documento XML con una sequenza di chiamate del metodo println, ma ciò sarebbe noioso, e dovreste anche cercare manualmente in ciascuna stringa i caratteri come < e sostituirli con le loro codifiche. Invece di fare così, dovreste costruire un oggetto di tipo Document e scriverlo in un flusso, usando una classe come Transformer.

Per costruire un oggetto di tipo Document a partire da un oggetto di una classe qualsiasi SomeClass, dovreste realizzare una classe come questa: public class MyBuilder

```
{
public Document build(SomeClass x)
```

...
}

I passi seguenti sono l'inverso dei passi che avete visto in Consigli pratici 2.

38

XML

Passo 1. *Esaminate il formato del documento XML che volete creare Come nel Passo 1 dei Consigli pratici 2, considerate tutti gli elementi che hanno altri elementi come contenuto, e identificate classi Java che vi corrispondano.*

Passo 2. *Per ciascuna classe identificata al Passo 1, aggiungete un metodo ausiliario alla classe del costruttore*

Il metodo ausiliario dovrebbe avere una struttura come questa: private Element createNomeElemento(ClasseElemento x)

{
Element e = doc.createElement("nome elemento");
Element child1 = ...;
e.appendChild(child1);
Element child2 = ...;
e.appendChild(child2);
...
return e;
}

Nella creazione di un elemento figlio, esistono due casi distinti. Se l'elemento figlio contiene testo, allora dovete creare un nodo di testo e un nodo figlio: Text childText = doc.createTextNode(testo);

Element child = doc.createElement("nome elemento figlio"); child.appendChild(childText);

Se, invece, l'elemento figlio non contiene testo, allora chiedete all'oggetto x quali sono i dati che devono far parte dell'elemento figlio e invocate il suo metodo createNomeElementoFiglio:

ClasseElementoFiglio y = x.getClasseElementoFiglio(); Element child = createNomeElementoFiglio(y);

Ad esempio, ecco come appare il metodo createItem per l'elemento item: private Element createItem(Item x)

{
Element e = doc.createElement("item");
Element quantityChild = doc.createElement("quantity"); Text quantityText =
doc.createTextNode("" + x.getQuantity());
quantityChild.appendChild(quantityText);
e.appendChild(quantityChild);
Product y = x.getProduct();
Element productChild = createProduct(y);
e.appendChild(productChild);
return e;
}
XML
39
Complessivamente, la classe InvoiceBuilder richiede cinque metodi ausiliari: public class InvoiceBuilder
{
...
private Element createInvoice(Invoice x) {...}
private Element createAddress(Address x) {...}
private Element createItems(ArrayList x) {...}
private Element createItem(Item x) {...}
private Element createProduct(Product x) {...}
}
Può darsi che troviate utile implementare questi metodi con un metodo "bottom up", iniziando dal metodo più semplice (come createProduct) e terminando con il metodo che crea l'elemento radice, createInvoice.
Passo 3. *Terminate il costruttore di documenti scrivendo un costruttore per la classe e il metodo build*
public class MyBuilder
{
public MyBuilder()
throws ParserConfigurationException
{
DocumentBuilderFactory factory
= DocumentBuilderFactory.newInstance();
builder = factory.newDocumentBuilder();

```
}
public Document build(ClasseElementoRadice x)
{
doc = builder.newDocument();
Element root = createNomeElementoRadice(x);
doc.appendChild(root);
return doc;
}
// metodi ausiliari
...
private DocumentBuilder builder;
private Document doc;
}
```

Ad esempio, nella seconda riga del metodo build della classe InvoiceBuilder, invocate

```
Element root = createInvoice(x);
```

Passo 4. *Usate una classe come Transformer per scrivere il Document in un flusso di uscita*

Ad esempio:

```
Invoice x = ...;
InvoiceBuilder builder = new InvoiceBuilder();
Document doc = builder.build(x);
Transformer t =
TransformerFactory.newInstance().newTransformer();
t.transform(new DOMSource(doc),
new StreamResult(System.out));
```

4

Definizione di tipi di documento

Nei due paragrafi precedenti, avete visto come leggere e scrivere documenti XML; in questo paragrafo imparerete a produrre definizioni di tipi di documento (DTD, Document Type Definition), che sono insiemi di regole per documenti di un tipo particolare che siano correttamente composti.

Ad esempio, considerate un tipo di documento items. Intuitivamente, items descrive una sequenza di elementi di tipo item, ciascuno dei quali contie-Un DTD è una sequenza di rego-

ne un elemento di tipo product e uno di tipo quantity. Un elemento di le che descrive quali siano gli ele-tipo product contiene, poi, un elemento di tipo description e uno di menti figli e gli attributi validi di

tipo price, e ciascuno di questi ultimi elementi contiene testo. Lo scopo ciascun tipo di elemento.

di un DTD è rendere formale questa descrizione.

Un DTD è una sequenza di regole che descrive:

■ Gli attributi validi per ciascun tipo di elemento

■ Gli elementi figli validi per ciascun tipo di elemento Per prima cosa, poniamo l'attenzione sugli elementi figli. Gli elementi figli validi di un elemento vengono descritti da una regola ELEMENT:

<!ELEMENT items (item)>*

Ciò significa che un elenco di articoli deve contenere una sequenza di zero o più elementi di tipo articolo.

Come potete vedere, la regola è delimitata da <!...> e contiene il nome dell'elemento (items) di cui si stanno definendo rigidamente i figli, seguito da una descrizione di quali figli siano consentiti.

Passiamo ora alla definizione di un nodo di tipo item:

<!ELEMENT item (product, quantity)>

Questo significa che i figli di un nodo di tipo item devono essere un nodo di tipo product, seguito da un nodo di tipo quantity.

La definizione di un prodotto è simile:

<!ELEMENT product (description, price)>

Infine, ecco le definizioni dei tre tipi di nodi rimanenti:

<!ELEMENT quantity (#PCDATA)>

<!ELEMENT description (#PCDATA)>

<!ELEMENT price (#PCDATA)>

Il simbolo #PCDATA si riferisce al testo, chiamato nella terminologia XML "Parsable Character DATA" (dati analizzabili di tipo carattere). I dati di tipo carattere possono contenere qualsiasi carattere, ma alcuni caratteri, come < e &, hanno un significato speciale in XML e devono essere codificati se

compaiono in dati di tipo carattere. La Tabella 1 mostra le codifiche per i caratteri speciali.

Tabella 1

Codifiche per caratteri speciali

Carattere

Codifica

Descrizione

<

<

Minore di (less than) o parentesi angolare aperta

>

>

Maggiore di (greater than) o parentesi angolare chiusa

&

&

Lettera e commerciale (ampersand)

‘

'

Apostrofo

“

"

Virgolette (quotation)

Tabella 2

Espressioni canoniche per il contenuto di elementi

Descrizione della regola

Contenuto dell’elemento

EMPTY

Non sono ammessi figli

(E)*

Qualsiasi sequenza di zero o più elementi E

(E+)

Qualsiasi sequenza di uno o più elementi E

(E?)

Elemento E facoltativo (zero o un elemento E) (E1, E2, …)

Elemento E1, seguito da E2 o …

(E1 | E2 | …)

Elemento E1 o E2 o …

(#PCDATA)

Solo testo

*(#PCDATA | E1 | E2 | ...)**

Qualsiasi sequenza di testo e di elementi E1, E2,..., in qualsiasi ordine

ANY

Qualsiasi figlio è ammesso, in qualsiasi quantità Il DTD completo per un elenco di articoli ha sei regole, una per ciascun tipo di elemento:

<!ELEMENT items (item)>*

<!ELEMENT item (product, quantity)>

<!ELEMENT product (description, price)>

<!ELEMENT quantity (#PCDATA)>

<!ELEMENT description (#PCDATA)>

<!ELEMENT price (#PCDATA)>

Osserviamo più attentamente le descrizioni dei figli consentiti. La Tabella 2 mostra le espressioni che potete usare per descrivere i figli di un elemento. La parola chiave EMPTY ha un significato evidente: un elemento dichiarato EMPTY non può avere figli. Ad esempio, il DTD che descrive il linguaggio HTML definisce l'elemento img come EMPTY: un'immagine ha soltanto attributi, che specificano il file che contiene l'immagine, la sua dimensione e posizionamento, ma non ha figli.

Regole più interessanti per definire i figli si possono comporre con le operazioni delle espressioni canoniche (+ ? , |), che potete vedere nella Tabella 2 e nella Figura 9. Avete già visto gli operatori * ("zero o più") e , (sequenza): i figli di un elemento di tipo items sono zero o più elementi di tipo item, mentre i figli di un elemento di tipo item sono una sequenza di elementi di tipo product e description.*

Potete anche combinare queste operazioni per comporre espressioni più complesse. Ad esempio:

<!ELEMENT section (title, (paragraph | (image, title?))+)> definisce un elemento di tipo section i cui figli sono: 1. Un elemento di tipo title

2. Una sequenza di uno o più gruppi come quello seguente:

• elementi di tipo paragraph

• elementi di tipo image, ciascuno seguito da un elemento facoltativo di tipo title ***Figura 9***

Operatori
per espressioni
canoniche nei DTD
XML
43

Quindi
<section>
<title/>
<paragraph/>
<image/>
<title/>
<paragraph/>
</section>
è lecito, mentre
<section>
<paragraph/>
<paragraph/>
<title/>
</section>

non lo è: non c'è un titolo iniziale, e il titolo finale non segue un'immagine. Fortunatamente, regole così complicate non sono, in realtà, molto frequenti.

Avete già visto la regola (#PCDATA): significa che i figli possono essere composti da qualsiasi sequenza di caratteri. Ad esempio, nel DTD del nostro elenco di prodotti, l'elemento product può contenere al suo interno qualsiasi carattere.

Potete anche consentire contenuto misto: sequenze di qualsiasi carattere ed elementi specifici. Tuttavia, nel contenuto misto perdete il controllo sull'ordine in cui compaiono gli elementi. Come spiegato in Consigli per la Qualità 2, dovreste evitare il contenuto misto per i DTD che descrivono insiemi di dati: questa caratteristica è stata pensata per documenti che contengono sia testo sia marcatori, come le pagine HTML.

Infine,potete consentire a un nodo di avere figli di qualsiasi tipo, anche se dovreste evitarlo in DTD che descrivono insiemi di dati.

Ora sapete come specificare quali possono essere i figli di un nodo, ma un DTD vi consente anche di controllare quali attributi siano consentiti per un elemento. La descrizione di un attributo è cosiffatta:

<!ATTLIST Elemento Attributo Tipo ValorePredefinito> Le descrizioni dei tipi di attributo più utili sono elencate nella Tabella 3. Il tipo CDATA descrive una sequenza di dati di tipo carattere. Come per #PCDATA, alcuni caratteri, quali < e &, devono essere codificati (<, & e così via). Non esiste alcuna differenza pratica fra i tipi CDATA e #PCDATA: semplicemente, usate CDATA nelle dichiarazioni di attributi e #PCDATA nelle dichiarazioni di elementi.

Tabella 3

Tipi di attributo più usati

Descrizione del tipo

Tipo di attributo

CDATA

Una sequenza di caratteri qualsiasi

(N | N | ...)

Uno tra N , N , ...

1

2

1

2

44

XML

Tabella 4

Impostazioni predefinite per gli attributi

Dichiarazione

Descrizione

#REQUIRED

L'attributo è necessario

#IMPLIED

L'attributo è facoltativo

N

Valore predefinito, da usare se l'attributo non viene specificato

#FIXED N

Se l’attributo viene specificato, deve contenere questo valore Invece di consentire valori arbitrari per gli attributi, potete specificare un numero finito di alternative. Ad esempio, può darsi che vogliate restringere l’insieme di varia-bilità dell’attributo currency, in modo che si possano specificare soltanto dollari sta-tunitensi, euro e yen giapponesi. Per farlo, usate la dichiarazione seguente:

<!ATTLIST price currency (USD | EUR | JPY) #REQUIRED> Ciascuna scelta deve essere composta di lettere, di cifre o dei caratteri speciali

- _ .

Esistono altre descrizioni di tipo che sono meno usate e che potete trovare nel manuale di riferimento del linguaggio XML [1].

La descrizione di un tipo di attributo è seguita da una dichiarazione predefinita (“di default”). Le parole chiave che possono comparire in una dichiarazione predefinita sono elencate nella Tabella 4.

Ad esempio, questa dichiarazione di attributo afferma che ciascun elemento di tipo price deve avere un attributo currency, il cui valore può essere una sequenza qualsiasi di caratteri:

<!ATTLIST price currency CDATA #REQUIRED>

Per soddisfare questa dichiarazione, ciascun elemento di tipo price deve avere un attributo currency, come <price currency=”USD”>. Non sarebbe lecito specificare un prezzo senza un’indicazione della valuta.

Per un attributo facoltativo, usate invece la parola chiave #IMPLIED.

<!ATTLIST price currency CDATA #IMPLIED>

Ciò significa che potete fornire un attributo che descrive la valuta in un elemento di tipo price, oppure che lo potete omettere. Se lo omettete, l’applicazione che elabora i dati XML assumerà implicitamente una valuta predefinita.

Una scelta migliore sarebbe quella di fornire esplicitamente il valore predefinito:

<!ATTLIST price currency CDATA “USD”>

Questo significa che l’attributo currency viene considerato come se avesse il valore USD nel caso in cui l’attributo stesso

non venga specificato. Se l'attributo non viene specificato, un parser XML segnalerà che il valore di currency è USD.

XML

45

Infine, potete stabilire che un attributo può soltanto essere identico a un determi-nato valore. Ad esempio, la regola

<!ATTLIST price currency CDATA #FIXED "USD"> significa che un elemento di tipo price deve essere privo dell'attributo currency (nel qual caso il parser XML segnalerà che il valore di currency è USD) oppure deve contenere l'attributo con il valore USD. Questo tipo di regola, evidentemente, non è molto frequente.

Avete così visto i costrutti più comuni utilizzati nei DTD: usandoli, potete definire i vostri DTD per documenti XML che descrivano insiemi di dati. Nel prossimo paragrafo, vedrete come specificare quale DTD deve essere utilizzato da un certo documento XML, e come fare in modo che un parser XML verifichi se un documento è conforme al suo DTD.

Non potete, ovviamente, inserire testo ed elementi dove vi pare: le regole nel DTD

indicano quali elementi possono essere presenti in un certo punto del documento. Ad esempio, il DTD per gli elenchi di prodotti, che svilupperemo nel seguito di questo capitolo, dirà che l'elemento di tipo price può figurare all'interno di un elemento di tipo product, ma non all'interno di un elemento di tipo item.

Consigli pratici 4

Scrivere un DTD

Si scrive un DTD per descrivere un insieme di documenti XML dello stesso tipo. Il DTD specifica quali elementi contengono elementi figli (e l'ordine in cui essi compaiono) e quali elementi contengono testo. Inoltre, specifica quali elementi possono avere attributi, quali attributi sono obbligatori, e quali valori predefiniti vengono usati in caso di attributi mancanti.

Queste regole riguardano i DTD che descrivono i dati di programmi, mentre i DTD

che descrivono veri e propri documenti hanno, generalmente, una struttura molto più complessa.

Passo 1. *Recuperate o scrivete un paio di documenti XML di esempio* ***Passo 2.*** *Fate un elenco di tutti gli elementi che figurano nel documento XML*

Passo 3. *Per ciascun elemento, decidete se i suoi figli sono elementi o testo In base ai Consigli per la Qualità 2, è meglio evitare elementi i cui figli siano una combinazione di entrambe le cose.*

Passo 4. *Per gli elementi che contengono testo, la regola nel DTD è la seguente:*

<!ELEMENT elemento nome (#PCDATA)>

Passo 5. *Per ciascun elemento che contiene altri elementi, fate un elenco dei possibili elementi figli*

Passo 6. *Per ciascuno di tali elementi, decidete in quale ordine e quante volte debbano figurare gli elementi figli*

Successivamente, componete la regola:

<!ELEMENT elemento nome figlio1 conteggio1, figlio2 conteggio2, ...> dove ciascun conteggio è una delle espressioni seguenti: ***Quantità Conteggio***

0 o 1

?

1

niente

0 o più

***

1 o più

+

Passo 7. *Decidete quali elementi devono avere attributi Rispettando i Consigli per la Qualità 1, è meglio evitare del tutto gli attributi, oppure limitarne al massimo l'utilizzo.*

Proviamo a eseguire questi passi per creare un DTD per documenti XML che descrivono una fattura.

Il Passo 1 richiede un documento XML di esempio: ne potete trovare uno in Consigli Pratici 1.

Il Passo 2 ci chiede un elenco di tutti gli elementi. Eccoli:

- *invoice*
- *number*

- *address*
- *name*
- *company*
- *street*
- *city*
- *state*
- *zip*
- *items*
- *item*
- *product*
- *description*
- *quantity*

Il Passo 3 ci chiede di separare gli elementi che contengono altri elementi da quelli che contengono testo. I seguenti elementi hanno altri elementi come contenuto:

- *invoice*
- *address*
- *items*
- *item*
- *product*

I rimanenti contengono testo.

Il passo 4 dice che gli elementi che contengono testo si possono gestire semplicemente, in questo modo:

```
<!ELEMENT number (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

Il Passo 5 ci chiede di elencare gli elementi figli per ciascuno degli elementi rimanenti:

- *invoice*
- *quantity*

- *address*
- *items*
- *address*
- *name*
- *company*
- *street*
- *city*
- *state*
- *zip*
- *items*
- *item*
- *item*
- *product*
- *quantity*
- *product*
- *description*
- *price*

Nel Passo 6, dovete ora decidere quali espressioni canoniche usare per combinare questi elementi figli. L'elemento di tipo items può contenere un numero qualsiasi di articoli, per cui la regola è

<!ELEMENT items (item)>*

Nei casi rimanenti, ciascun elemento figlio può figurare una e una sola volta, generando le regole seguenti:

<!ELEMENT invoice (number,address,items)>

<!ELEMENT address (name,company,street,city,state,zip)>

<!ELEMENT item (product,quantity)>

<!ELEMENT product (description,price)>

Il Passo 7 suggerisce di non aggiungere attributi a meno che non vi sia una buona ragione: dato che non c'è, il DTD è completo.

Argomenti avanzati 1

La specifica di uno schema XML

Un DTD ha un limite: non potete specificare con precisione quale contenuto deve avere ciascun elemento XML. Ad esempio, non potete costringere un elemento a contenere

soltanto un numero o una data: un elemento di tipo (#PCDATA) può contenere qualsiasi stringa di testo. La specifica di uno schema XML supera queste limitazioni.

Uno schema XML è come un DTD, in quanto è un insieme di regole che devono essere rispettate da documenti di un particolare tipo, ma può contenere descrizioni di regole molto più precise.

Qui trovate soltanto un accenno di come si possa specificare uno schema XML. Per ciascun elemento, se ne specifica il nome e il tipo: i nomi di elementi hanno lo stesso significato di quelli presenti in un DTD, mentre i tipi sono una novità. Esistono molti tipi di base, alcuni dei quali sono elencati nella Tabella 5.

Tabella 5

Un sottoinsieme dei tipi di base di uno schema XML

Tipo

Significato

string

Una stringa qualsiasi

integer

Un numero intero

double

Un numero in virgola mobile

boolean

false o true

date

Una data, come 2001-01-23

time

Un orario, come 20:15

Ecco una definizione tipica:

<xsd:element name=”quantity” type=”xsd:integer”/> XML

49

Questa dichiarazione definisce l’elemento quantity come un elemento che contiene testo, nel quale il testo non può avere un valore qualsiasi, ma deve essere un numero intero. Il prefisso xsd: è il prefisso di uno spazio di nomi, che dichiara che xsd:element e xsd:integer fanno parte dello spazio dei nomi della definizione degli schemi XML

(XSD, XML Schema Definition). Consultate Argomenti Avanzati 2 per avere maggiori informazioni sugli spazi di nomi.

Potete anche definire tipi complessi, più o meno come si definiscono le classi in Java. Ecco la definizione di un tipo Address:

```
<xsd:complexType name="Address">
<xsd:sequence>
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="company" type="xsd:string"/>
<xsd:element name="street" type="xsd:string"/>
<xsd:element name="city" type="xsd:string"/>
<xsd:element name="state" type="xsd:string"/>
<xsd:element name="zip" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
```

Per indicare che un elemento può essere ripetuto, usate minOccurs e maxOccurs:

```
<xsd:complexType name="ItemList">
<xsd:element name="item" type="Item"
minOccurs="0" maxOccurs="unbounded"/>
</xsd:complexType>
```

A questo punto siete in grado di specificare che una fattura deve avere i campi shipto e billto, entrambi di tipo Address:

```
<xsd:element name="shipto" type="Address"/>
<xsd:element name="billto" type="Address"/>
```

Questi esempi mostrano che uno schema XML può essere più semplice e più preciso di un DTD. Per questa ragione, è probabile che, nel lungo termine, i DTD assumano minore importanza e le definizioni di schemi XML diventino il metodo preferito per descrivere tipi di documenti XML.

La specifica degli schemi XML ha molte caratteristiche avanzate: consultate il sito Web dell'organizzazione W3C [3] per avere maggiore dettagli.

5

Analisi sintattica con definizioni di tipi di documento

Fino a ora avete visto come effettuare l'analisi sintattica di un documento XML senza usare un DTD, ma è sempre una

buona idea inserire in ciascun documento XML il riferimento a un DTD e fare in modo che il parser verifichi che il documento ne segua le regole. In questo modo, il parser può verificare che nel documento non siano pre-50

XML

senti errori. Ad esempio, se il parser sa che i figli di un elemento sono, a loro volta, elementi, allora può eliminare le spaziature e potete risparmiare il tempo che perdere-ste a cercarle ed eliminarle.

Un documento XML può conte-

Nel paragrafo precedente avete visto come sviluppare un DTD per nere il proprio DTD oppure fare

una classe di documenti XML, specificando in esso gli elementi am-riferimento a un DTD memoriz-

messi nel documento e i loro attributi. Un documento XML può fare zato altrove.

riferimento a un DTD i due modi:

1. Il documento può contenere il DTD.

2. Il documento può fare riferimento a un DTD memorizzato altrove.

Un DTD viene inserito con la dichiarazione DOCTYPE. Se il documento contiene il proprio DTD, la dichiarazione è:

<!DOCTYPE elementoRadice [regole]>

Ad esempio, un elenco di articoli può contenere il proprio DTD, in questo modo:

```
<?xml version="1.0"?>
<!DOCTYPE items [
<!ELEMENT items (item*)>
<!ELEMENT item (product, quantity)>
<!ELEMENT product (description, price)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT price (#PCDATA)>
]>
<items>
<item>
<product>
<description>Ink Jet Refill Kit</description>
```

<price>29.95</price>
</product>
<quantity>8</quantity>
</item>
<item>
<product>
<description>4-port Mini Hub</description>
<price>19.95</price>
</product>
<quantity>4</quantity>
</item>
</items>

Facendo riferimento a un DTD

Ma, se il DTD è più complesso, è meglio memorizzarlo al di fuori del esterno, dovete fornire il nome

documento XML. In tal caso, usate la parola chiave SYSTEM all'interno di un file o uno URL.

della dichiarazione DOCTYPE, per indicare che il DTD deve essere ricer-cato dal sistema che ospita l'elaboratore XML. La parola chiave SYSTEM

è seguita dall'identificazione del DTD, che può essere un file locale:

<!DOCTYPE items SYSTEM "items.dtd">

XML

51

oppure un risorsa sul Web, caratterizzata da uno URL:

<!DOCTYPE items

SYSTEM "http://www.mycompany.com/dtds/items.dtd"> Nel caso di DTD di utilizzo generale, la dichiarazione DOCTYPE può contenere la parola chiave PUBLIC, come in questo esempio:

<!DOCTYPE Web-app

PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd"> Quando il vostro documento XML

Un programma che analizzi la descrizione del DTD può esaminare ha un DTD, abilitate la verifica sin-l'identificativo

pubblico: se gli è familiare, può evitare di recuperare il tattica nel parser.

DTD dallo URL.

Quando includete un DTD in un documento XML, dovreste indicare al parser di verificare sintatticamente il documento: in questo caso il parser controllerà che tutti gli elementi figli e gli attributi di un elemento siano conformi alle regole ELEMENT e ATTLIST presenti nel DTD. Se un documento non è valido, il parser lancerà un'eccezione. Per abilitare la verifica sintattica, usate il metodo setValidating della classe DocumentBuilderFactory, prima di invocare il metodo newDocumentBuilder: DocumentBuilderFactory factory =

DocumentBuilderFactory.newInstance();

factory.setValidating(true);

DocumentBuilder builder = factory.newDocumentBuilder(); Document doc = builder.parse(...);

Uno dei grandi vantaggi della verifica sintattica consiste nella certezza della correttezza sintattica del documento, una volta che sia stato letto e verificato. Ad esempio, se il DTD specifica che gli elementi figli di ciascun elemento di tipo item devono essere gli elementi product e quantity, in questo ordine, allora potete basarvi su tale ipotesi e non avete bisogno di inserire noiosi controlli nel vostro codice. Confrontate la classe ItemListParser al termine di questo paragrafo con quella del Paragrafo 2, osservando quanto sia più semplice il codice.

Quando analizzate un documen-

Se il parser può utilizzare il DTD associato al documento, può appli-to XML avente un DTD, segnala-

care anche un altro miglioramento. Ricordate che nel precedente prote al parser di ignorare le spazia-

gramma di esempio dovevamo eliminare gli spazi restituiti dal parser: ture.

in quel programma, il parser non utilizzava il DTD, per cui non aveva modo di sapere se gli spazi fossero significativi oppure no, ma, se il parser conosce il DTD, può eliminare automaticamente gli spazi in quegli elementi che non hanno testi di tipo #PCDATA come figli (cioè, quegli elementi che hanno altri elementi come contenuto). Ad esempio, nel DTD

dell'elenco di prodotti, i figli di un elemento di tipo product sono tutti elementi, non c'è testo. Perciò, qualsiasi carattere di spaziatura presente tra gli elementi può essere ignorato. Per generare parser che abbiano tale comportamento, dovete invocare il metodo setIgnoringElementContentWhitespace della classe DocumentBuilderFactory.

factory.setValidating(true);

factory.setIgnoringElementContentWhitespace(true); ***52***

XML

Infine, se il parser può utilizzare il DTD, può anche inserire valori predefiniti negli attributi. Ad esempio, supponete che un DTD definisca un attributo currency per l'elemento di tipo price:

<!ATTLIST price currency CDATA "USD">

Se un documento contiene un elemento di tipo price senza l'attributo currency, il parser può fornire il valore predefinito:

String attributeValue

= priceElement.getAttribute("currency");

// la stringa vale "USD" se l'attributo non è stato specificato Come evidenziato da queste proprietà, costruire un DTD e fornirlo insieme a tutti i documenti XML è sempre una buona idea.

Ciò conclude la nostra discussione sul linguaggio XML: a questo punto conoscete abbastanza bene XML per metterlo all'opera nella descrizione del formato dei dati.

Ogni volta che avete la tentazione di usare un formato di file "veloce e sporco", dovreste, invece, considerare un possibile utilizzo di XML. Usando il linguaggio XML per lo scambio dei dati, i vostri programmi diventano più professionali, robusti e flessibili.

Questo capitolo ha trattato gli aspetti più importanti del linguaggio XML, relativi alla programmazione quotidiana; per le caratteristiche più avanzate, che possono essere utili in situazioni particolari, consultate [1]. Inoltre, la tecnologia XML sta ancora su-bendo un rapido sviluppo, per cui è bene tenersi aggiornati sull'argomento: [3] e [4]

sono buoni punti di riferimento sul Web.

File ItemListParser.java

import java.io.File;

```
import java.io.IOException;
import java.util.ArrayList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory; import
javax.xml.parsers.ParserConfigurationException; import
org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Text;
import org.xml.sax.SAXException;
/**
Un parser XML per elenchi di articoli.
*/
public class ItemListParser
{
/**
Costruisce un parser in grado di analizzare elenchi di articoli.
*/
XML
53
public ItemListParser()
throws ParserConfigurationException
{
DocumentBuilderFactory factory
= DocumentBuilderFactory.newInstance();
factory.setValidating(true);
factory.setIgnoringElementContentWhitespace(true); builder
= factory.newDocumentBuilder();
}
/**
Analizza un file XML contenente un elenco di articoli.
@param fileName il nome del file
@return un vettore contenente tutti gli articoli del file XML
*/
public ArrayList parse(String fileName)
```

```
throws SAXException, IOException
{
File f = new File(fileName);
Document doc = builder.parse(f);
// recupera l'elemento radice <items>
Element root = doc.getDocumentElement();
return getItems(root);
}
/**
Costruisce un vettore di articoli estratti da un elemento DOM.
@param e un elemento di tipo <items>
@return un vettore di tutti i figli di tipo <item> di e
*/
private static ArrayList getItems(Element e)
{
ArrayList items = new ArrayList();
// recupera i figli di tipo <item>
NodeList children = e.getChildNodes();
for (int i = 0; i < children.getLength(); i++)
{
Element childElement
= (Element)children.item(i);
Item c = getItem(childElement);
items.add(c);
}
return items;
}
/**
Costruisce un articolo estratto da un elemento DOM.
@param e un elemento di tipo <item>
@return l'articolo descritto dall'elemento
*/
```



```
private static Item getItem(Element e)
{
NodeList children = e.getChildNodes();
```

Product p = getProduct((Element)children.item(0)); Element quantityElement
= (Element)children.item(1);
Text quantityText
= (Text)quantityElement.getFirstChild();
int quantity
= Integer.parseInt(quantityText.getData());
return new Item(p, quantity);
}
*/**
Costruisce un prodotto estratto da un elemento DOM.
@param e un elemento di tipo <product>
@return il prodotto descritto dall'elemento
*/
private static Item getProduct(Element e)
{
NodeList children = e.getChildNodes();
Element descriptionElement
= (Element)children.item(0);
Text descriptionText
= (Text)descriptionElement.getFirstChild();
String description = descriptionText.getData();
Element priceElement = (Element)children.item(1); Text priceText
= (Text)priceElement.getFirstChild();
double price
= Double.parseDouble(priceText.getData());
return new Product(description, price);
}
private DocumentBuilder builder;
}*

File ItemListParserTest.java

*import java.util.ArrayList;
/**
Questo programma analizza un file XML contenente un elenco di articoli.
Il file XML dovrebbe fare riferimento al file items.dtd.
/

public class ItemListParserTest
{
public static void main(String[] args)
XML
55
throws Exception
{
ItemListParser parser = new ItemListParser();
ArrayList items = parser.parse("items.xml");
for (int i = 0; i < items.size(); i++)
{
Item anItem = (Item)items.get(i);
System.out.println(anItem.format());
}
}
}
Consigli per la produttività 1
Usate i DTD con i vostri file XML

Sviluppare un DTD per un insieme di file XML è uno sforzo aggiuntivo, ma tale sforzo viene ripagato quasi immediatamente, perché il codice che traduce un documento in un oggetto Java diventa molto più semplice. Ad esempio, confrontate i metodi getItem del parser del Paragrafo 2 (che non usa un DTD) e del parser del Paragrafo 5 (che, invece, verifica sintatticamente un DTD e ignora le spaziature). La prima versione non sa in che ordine si presentano gli elementi, e deve ignorare manualmente i nodi contenenti spaziature; ciò rende il metodo abbastanza noioso da realizzare: private static Item getItem(Element e)

{
NodeList children = e.getChildNodes();
Product p = null;
int quantity = 0;
for (int j = 0; j < children.getLength(); j++)
{
Node childNode = children.item(j);
if (childNode instanceof Element)
{

Element childElement = (Element)childNode;
String tagName = childElement.getTagName();
if (tagName.equals("product"))
p = getProduct(childElement);
else if (tagName.equals("quantity"))
{
Text textNode =
= (Text)childElement.getFirstChild();
String data = textNode.getData();
quantity = Integer.parseInt(data);
}
}
}
return new Item(p, quantity);
}

56

XML

La seconda versione è molto più breve: segue semplicemente la regola del DTD

<!ELEMENT item (product, quantity)>

e, quindi, sa che un elemento di tipo item ha due figli, product e quantity.

private static Item getItem(Element e)
{
NodeList children = e.getChildNodes();
Product p = getProduct((Element)children.item(0)); Element quantityElement
= (Element)children.item(1);
Text quantityText
= (Text)quantityElement.getFirstChild();
int quantity
= Integer.parseInt(quantityText.getData());
return new Item(p, quantity);
}

Se c'è un errore nel file XML, il parser si rifiuterà di analizzarlo: il vostro codice non si deve preoccupare di dati errati in ingresso, un gran risparmio di tempo.

Errori comuni 2

Non potete leggere da un flusso
un documento XML avente un DTD esterno

Se il vostro documento si riferisce a un DTD esterno, come in questo esempio

<!DOCTYPE productlist SYSTEM "dtds/productlist.dtd"> allora non potete leggere il documento tramite un oggetto di tipo InputStream.

InputStream in = ...;

// un flusso di ingresso contenente il documento XML

Document doc = builder.parse(in);

// non funziona con un DTD esterno

Dovete, invece, leggere il documento da un file o da uno URL; questo funzionerà: File f = ...;

Document doc = builder.parse(f);

Il DTD viene, in questo caso, caricato facendo riferimento alla cartella che contiene il file, ma se leggete da un flusso, il parser non ha idea di come trovare il DTD. Dopo tutto, un flusso è soltanto una sequenza di byte, che potrebbero provenire da qualsiasi fonte.

Altre tecnologie XML

Questo capitolo tratta il sottoinsieme della specifica XML 1.0 che è più utile per le situazioni di programmazione più frequenti. Fin dal momento in cui la versione 1.0

della specifica XML è stata pubblicata, c'è stato un enorme interesse nelle tecnologie XML più avanzate, tra le quali stanno emergendo alcuni utili sviluppi:

- *definizioni di schemi*
- *spazi dei nomi*
- *XHTML*
- *XSL e trasformazioni*

Gli Argomento avanzati 1 contengono ulteriori informazioni sulle definizioni di schemi.

Gli spazi di nomi sono stati progettati per essere certi che diverse persone e organizzazioni possano sviluppare documenti XML senza entrare in conflitto con i nomi degli elementi. Ad esempio, se leggete gli Argomenti Avanzati 1, vedrete che le definizioni di schemi XML hanno nomi di elementi con un prefisso xsd:, in questo modo:

<xsd:element name="city" type="xsd:string"/> Di conseguenza, i nomi di marcatori e di attributi come element e string non entrano in conflitto con altri nomi. A questo proposito, gli spazi di nomi sono simili ai pacchetti Java. Tuttavia, un prefisso che indica uno spazio di nomi, come xsd:, è soltanto un'abbreviazione per il vero identificatore dello spazio di nomi, che è una stringa univoca, molto più lunga. Lo spazio dei nomi completo per le definizioni di schemi XML è http://www.w3.org/2000/08/XMLSchema. Ciascuna definizione di schema inizia con l'enunciato

<xsd:schema xmlns:xsd="http://www.w3.org/2000/08/XMLSchema"> che collega il prefisso xsd allo spazio di nomi completo.

XHTML è la raccomandazione più recente dell'organizzazione W3C per la scrittura delle pagine Web. Diversamente dal linguaggio HTML, il linguaggio XHTML è to-talmente conforme allo standard XML. Una volta che gli strumenti di scrittura per il Web passeranno a XHTML, diventerà molto più facile scrivere programmi che analiz-zino pagine Web; inoltre, lo standard XHTML è stato

accuratamente progettato per essere compatibile con i broswer esistenti.

Mentre i documenti XHTML sono destinati a essere visualizzati in un browser, i documenti XML, in generale, non sono affatto progettati per essere visualizzati. Nonostante ciò, spesso si vuole trasformare un documento XML in un formato visualiz-zabile: il linguaggio XSL (Extensible Style Language) è stato creato proprio per questo motivo. Un foglio di stile indica come trasformare un documento XML in un documento HTML, oppure anche in un formato completamente diverso, come PDF.

Per maggiori informazioni su queste e altre tecnologie emergenti, consultate il sito Web dell'organizzazione W3C [3].

Riepilogo del capitolo

1. Il linguaggio XML consente la codifica di dati complessi, indipendentemente da qualsiasi linguaggio di programmazione, in una forma che il destinatario può analizzare facilmente.

2. I file XML sono leggibili sia dai programmi per computer sia dalle persone.

3. I file XML sono leggibili sia dai programmi per computer sia dagli umani.

4. Sia HTML che XML sono derivati da SGML, Standard Generalized Markup Language.

5. XML descrive il significato dei dati, non come visualizzarli.

6. Un insieme di dati XML viene detto documento: inizia con un'intestazione (header) e contiene elementi e testo.

7. Un elemento può contenere testo, sottoelementi, oppure una loro combinazione ("contenuto misto"). Nelle descrizioni di dati, evitate il contenuto misto.

8. Gli elementi possono avere attributi: usateli per descrivere come interpretare il contenuto degli elementi.

9. Un parser è un programma che legge un documento, verifica che sia sintatticamente corretto ed esegue determinate azioni mentre elabora il documento.

10. Esistono due tipi di parser: i parser SAX generano eventi mentre analizzano un documento, mentre i parser DOM costruiscono un albero.

11. Un oggetto di tipo DocumentBuilder è in grado di leggere un documento XML da un file, da uno URL o da un flusso di ingresso. Il risultato è un oggetto di tipo Document, che contiene un albero.

12. L'albero che rappresenta un documento è composto di nodi. I tipi di nodi più importanti sono Element e Text.

13. Per visitare i nodi figli di un elemento, usate un oggetto di tipo NodeList.

14. Se non usate un DTD, il parser XML conserva tutte le spaziature.

15. Per scandire gli attributi di un elemento, usate un oggetto di tipo NamedNodeMap.

Ciascun attributo è memorizzato in un oggetto di tipo Node.

16. L'interfaccia Document ha metodi per creare elementi e nodi di testo.

17. Per scrivere un documento XML in un flusso usate un oggetto di tipo Transformer.

18. Un DTD è una sequenza di regole che descrive quali siano gli elementi figli e gli attributi validi di ciascun tipo di elemento.

19. Un documento XML può contenere il proprio DTD oppure fare riferimento a un DTD memorizzato altrove.

20. Facendo riferimento a un DTD esterno, dovete fornire il nome di un file o uno URL.

21. Quando il vostro documento XML ha un DTD, abilitate la verifica sintattica nel parser.

22. Quando analizzate un documento XML avente un DTD, segnalate al parser di ignorare le spaziature.

Ulteriori letture

[1] http://www.xml.com/axml/axml.html Specifica XML commentata.

[2] James Gosling e altri, The Java Language Specification, 2nd edition, Addison-Wesley, 2000.

[3] http://www.w3c.org/xml Il sito Web dell'organizzazione W3C dedicato a XML

[4] http://java.sun.com/xml Il sito Web di Sun Microsystems dedicato a XML

Classi, oggetti e metodi
presentati nel capitolo

javax.xml.parsers.DocumentBuilder
newDocument
parse
javax.xml.parsers.DocumentBuilderFactory
newInstance
newDocumentBuilder
setIgnoreElementContentWhitespace
setValidating
javax.xml.transform.Transformer
transform
javax.xml.transform.TransformerFactory
newInstance
newTransformer
javax.xml.transform.dom.DOMSource
javax.xml.transform.stream.StreamResult
org.w3c.dom.Document
createElement
createTextNode
getDocumentElement
org.w3c.dom.Element
getAttribute
getAttributes
getTagName

setAttribute
org.w3c.dom.NamedNodeMap
getLength
item
60
XML
org.w3c.dom.Node
getChildNodes
getFirstChild
getLastChild
org.w3c.dom.NodeList
getLength
item
org.w3c.dom.Text
getData

Esercizi di ripasso

Esercizio R.1. *Fornite alcuni esempi che evidenzino le differenze tra i linguaggi XML e HTML.*

Esercizio R.2. *Progettate un documento XML che descriva un conto bancario.*

Esercizio R.3. *Disegnate una vista ad albero per il documento XML che avete creato nell'esercizio precedente.*

Esercizio R.4. *Scrivete il documento XML che corrisponde all'albero sintattico di Figura 6.*

Esercizio R.5. *Scrivete il documento XML che corrisponde all'albero sintattico di Figura 7.*

Esercizio R.6. *Create un documento XML che descriva un libro, con elementi figli per il nome dell'autore, il titolo e l'anno di pubblicazione.*

Esercizio R.7. *Aggiungete al documento dell'esercizio precedente la descrizione della lingua del libro. Usereste un elemento o un attributo?*

Esercizio R.8. *Cos'è un contenuto misto? Quali problemi provoca?*

Esercizio R.9. *Progettate un documento XML che descriva un borsellino contenente tre monete da un quarto di dollaro (quarter), un moneta da dieci centesimi (dime) e due monete da cinque centesimi (nickel).*

Esercizio R.10. *Spiegate perché un programma di disegno come Microsoft Paint è un programma WYSIWYG che è anche del tipo "ciò che vedete è quello che siete riusciti a fare".*

Esercizio R.11. *Come potete visitare i figli di un nodo senza usare un oggetto di tipo NodeList? Suggerimento: getNextSibling.*

Esercizio R.12. *Quali sono le differenze tra un oggetto di tipo NodeList e uno di tipo ArrayList?*

Esercizio R.13. *Quali sono le differenze tra un oggetto di tipo NodeList e uno di tipo LinkedList?*

Esercizio R.14. *Quali sono le differenze tra un oggetto di tipo NodeList e uno di tipo NamedNodeMap?*

Esercizio R.15. *Progettate un DTD che descriva una banca con i suoi conti bancari.*

Esercizio R.16. *Progettate un DTD che descriva un lettore di una biblioteca che ha preso a prestito un insieme di libri. Ciascun libro ha un numero di identificazione (ID), un autore e un titolo. Il lettore ha un nome e un numero di telefono.*

Esercizio R.17. *Scrivete il DTD per il seguente documento XML*

```
<?xml version="1.0">
<productlist>
<product>
<name>Comtrade Tornado</name>
<price currency="USD">2495</price>
<score>60</score>
</product>
<product>
```

<name>AMAX Powerstation 75</name>
<price>2999</price>
<score>62</score>
</product>
</productlist>

Esercizio R.18. *Progettate un DTD per descrivere fatture, come descritto in Consigli pratici 2.*

Esercizio R.19. *Progettate un DTD per descrivere semplici frasi in lingua inglese, come descritto in Note di cronaca 2.*

Esercizio R.20. *Progettate un DTD per descrivere espressioni aritmetiche, come descritto in Note di cronaca 2.*

Esercizi di programmazione

Esercizio P.1. *Scrivete un programma che sia in grado di leggere file XML come questo:*

<purse>
<coin>
<value>0.5</value>
<name>half dollar</name>
</coin>
...
</purse>

Il programma dovrebbe costruire un oggetto di tipo Purse e visualizzare il valore totale delle monete contenute nel borsellino.

Esercizio P.2. *Partendo dall'esercizio precedente, fate in modo che il programma possa leggere un file XML come già descritto, per poi visualizzare un file XML con il seguente formato:*

<purse>
<coins>
<coin>
<value>0.5</value>
<name>half dollar</name>

</coin>
<quantity>3</quantity>
</coins>
<coins>
<coin>
<value>0.25</value>
<name>quarter</name>
</coin>
<quantity>2</quantity>
</coins>
</purse>

Esercizio P.3. *Ripetete l'Esercizio P.1 usando un DTD per la verifica sintattica.*

Esercizio P.4. *Scrivete un programma che sia in grado di leggere file XML come questo:*

<bank>
<account>
<number>3</number>
<balance>1295.32</balance>
</account>
...
</bank>

Il programma dovrebbe costruire un oggetto di tipo Bank e visualizzare il valore totale dei saldi dei conti bancari.

Esercizio P.5. *Ripetete l'esercizio precedente usando un DTD per la verifica sintattica.*

Esercizio P.6. *Migliorate l'Esercizio P.4 come segue: per prima cosa, leggete il file XML; quindi, aggiungete il 10% di interessi a tutti i conti bancari; infine, scrivete un file XML che contenga i nuovi saldi dei conti correnti.*

Esercizio P.7. *Scrivete un programma che sia in grado di leggere file XML come questo:*

<rectangle>
<x>5</x>

<y>10</y>
<width>20</width>
<height>30</height>
</rectangle>
Tracciate la forma in una finestra.

Esercizio P.8. *Scrivete un programma che sia in grado di leggere file XML come questo:*

<ellipse>
<x>5</x>
<y>10</y>
<width>20</width>
<height>30</height>
</ellipse>
Tracciate la forma in una finestra.

Esercizio P.9. *Scrivete un programma che sia in grado di leggere file XML come questo:*

<rectangularshape shape="ellipse">
<x>5</x>
<y>10</y>
<width>20</width>
<height>30</height>
</rectangularshape>
e che accetti, come valori dell'attributo shape, le stringhe "rectangle", "diamond" e
"ellipse".
Tracciate la forma in una finestra.

Esercizio P.10. *Scrivete un programma che sia in grado di leggere file XML come questo:*

<drawing>
<rectangle>
<x>5</x>
<y>10</y>
<width>20</width>
<height>30</height>
</rectangle>
<line>

<x1>5</x1>
<y1>10</y1>
<x2>25</x2>
<y2>40</y2>
</line>
<message>
<text>Hello, World!</text>
<x>20</x>

<y>30</y>
</message>
</drawing>
Tracciate il disegno in una finestra.

Esercizio P.11. *Ripetete l'esercizio precedente usando un DTD per la verifica sintattica.*

Esercizio P.12. *Scrivete un programma che sia in grado di leggere file XML come questo:*

<polygon>
<point>
<x>5</x>
<y>10</y>
</point>
...
</polygon>

Tracciate la forma in una finestra.

Esercizio P.13. *Scrivete un DTD che descriva documenti contenenti informazioni relative a nazioni: il nome della nazione, la sua popolazione e la sua superficie. Create un file XML con cinque diverse nazioni. Il DTD e i dati XML devono trovarsi in file diversi. Scrivete un programma che usi il file XML che avete scritto e visualizzi:*

- *La nazione con la superficie maggiore*
- *La nazione con la popolazione maggiore*
- *La nazione con la densità più elevata (persone per chilometro quadrato)* ***Esercizio P.14.*** *Scrivete un parser per*

analizzare fatture, come descritto in Consigli pratici 1. Il parser deve creare dapprima una struttura ad albero che rappresenti la fattura, per poi visualizzarla in un formato commerciale.

Esercizio P.15. *Modificate l'esercizio precedente per consentire indirizzi di consegna e di fatturazione diversi.*

Esercizio P.16. *Scrivete un costruttore di documenti che trasformi un oggetto che rappresenta una fattura in un file XML avente il formato descritto in Consigli pratici 1.*

Esercizio P.17. *Modificate l'esercizio precedente per consentire indirizzi di consegna e di fatturazione diversi.*

Esercizio P.18. *Progettate un formato XML per descrivere gli appuntamenti in un agenda. Un appuntamento si compone di una data, di un orario di inizio, di un orario di fine e di una descrizione, per esempio:*

Dentista 1/10/2001 17:30 18:30

Corso CS1 22/10/2001 08:30 10:00

XML

Scrivete un parser per tale formato di file che generi un oggetto di tipo Appointment-Calendar contenente oggetti di tipo Appointment.

Esercizio P.19. *Migliorate l'esercizio precedente in modo che il programma legga dapprima un file di appuntamenti, poi un altro file con il seguente formato:*

```
<commands>
<add>
<appointment>
...
</appointment>
</add>
...
<remove>
<appointment>
...
</appointment>
```

</remove>

</commands>

Il programma deve elaborare i comandi e generare un file XML contenente gli appuntamenti aggiornati.

Esercizio P.20. *Scrivete un programma per simulare il sistema di prenotazione dei posti a sedere di una compagnia aerea, usando documenti XML. Supponete che l'ae-roplano abbia 20 posti in prima classe (5 file di 4 posti ciascuna, separati da un corridoio) e 180 posti in classe economica (30 file di 6 posti ciascuna, separati da un corridoio). Il programma legge un file che descrive la situazione di occupazione dei posti, in un formato XML di vostra scelta, e un file di comandi per aggiungere passeggeri, in un formato simile a quello dell'esercizio precedente. Ciascun comando specifica la classe (prima o economica), il numero di passeggeri che viaggiano assieme (1 o 2 in prima classe; da 1 a 3 in economica) e la posizione preferita (lato finestrino o corridoio in prima classe; corridoio, centro o finestrino in classe economica). Il programma elabora i comandi e produce come risultato un file con la situazione aggiornata dell'occupazione dei posti.*

JavaServer Pages

Obiettivi del capitolo

- *Realizzare pagine Web dinamiche con la tecnologia JavaServer Pages*
- *Apprendere gli elementi sintattici di JavaServer Pages*
- *Strutturare un'applicazione Web come sequenza di JavaServer Pages*
- *Capire la relazione tra JavaServer Pages e servlet* ***2***

JavaServer Pages

Quando un browser Web richiede una pagina HTML a un server Web, il server recupera il file e ne invia il contenuto al client. Il contenuto è statico: viene visualizzato lo stesso contenuto ogni volta che viene richiesta la pagina, finché il file HTML non viene sostituito da un altro file. Tuttavia, i siti Web più interessanti forniscono contenuti dinamici: ogni

volta che tornate a visitare un sito Web di notizie o di acquisti, ricevete informazioni diverse, spesso adattate alle vostre preferenze personali. In questo capitolo vedrete come realizzare contenuti dinamici per il Web usando JavaServer Pages e servlet.

1

Contenuto Web dinamico

Per usare la tecnologia JavaServer Pages (JSP) per produrre contenuto Web dinamico, avete bisogno di un server Web integrato con un contenitore JSP. Sono disponibili diversi contenitori JSP: un prodotto eccellente e disponibile gratuitamente è il server Apache Tomcat ([1]). Per provare i programmi di questo capitolo, faremo ora l'ipotesi che abbiate installato un server Web e un contenitore JSP.

File date.jsp

```
<html>
<head>
<title>Date JSP</title>
</head>
<body>
<h1>Date JSP</h1>
<p>The current time is:
<%= new java.util.Date() %>
</p>
</body>
</html>
```

Una pagina JavaServer Pages

Come potete vedere, una pagina JSP assomiglia molto a una normale (JSP) contiene marcatori HTML

pagina HTML, ma, all'interno del codice HTML, trovate l'istruzione: e istruzioni Java.

```
<%= new java.util.Date() %>
```

Questa istruzione viene eseguita ogni volta che la pagina Web viene inviata a un browser, e il valore dell'espressione viene inserita nella pagina. Ogni volta che richiedete questa

pagina, vengono visualizzati un diverso orario e una diversa data, come mostrato in Figura 1.

Per caricare la pagina nel server, eseguite questi passi: 1. Usate un editor di testo per scrivere il file JSP.

2. Inserite il file date.jsp in una cartella di applicazioni Web del vostro motore JSP. Ad esempio, se usate Tomcat, dovreste creare una sottocartella come c:\jakarta-tomcat\webapps\bigjava

JavaServer Pages

3

Figura 1

Esecuzione della pagina che visualizza la data

3. Eseguite il server Web.

4. Visualizzate la pagina nel vostro browser, usando, ad esempio, l'indirizzo localhost:8080/bigjava/date.jsp

Il contenitore JSP rimuove tutti i

Questo esempio mostra un costrutto importante di JavaServer Pages. Il marcatori JSP e inserisce al loro

contenitore JSP legge la pagina JSP che è stata richiesta e la trasforma in posto le stringhe prodotte dalla

una pagina HTML. I marcatori HTML standard non vengono toccati, valutazione delle espressioni Java.

mentre tutte le espressioni racchiuse tra i marcatori JSP

<%= … %>

vengono valutate e convertite in una stringa (usando il metodo toString). La stringa risultante viene inserita nella pagina HTML. Vedrete altri marcatori JSP più avanti, in questo capitolo: il contenitore JSP li rimuove e li elabora tutti, in modo che il documento risultante contenga soltanto codice HTML. Il server Web invia al browser tale documento (osservate la Figura 2).

Figura 2

Il contenitore JSP

riscrive la pagina

richiesta

2

Inserire le elaborazioni in oggetti JavaBean

Nell'esempio precedente, la parte dinamica della pagina JSP era molto semplice: un'unica espressione Java. Nella maggior parte dei casi, una pagina JSP deve svolgere molte elaborazioni per produrre un risultato interessante e, sebbene sarebbe, in teo-ria, possibile aggiungere ulteriore codice Java direttamente nella pagina JSP, raramen-te fare così è una buona idea, per un motivo molto semplice: la maggior parte delle pagine Web professionali vengono progettate da due diverse categorie di esperti, un programmatore che sa come calcolare i risultati da visualizzare nella pagina, e un progettista grafico che decide come visualizzare, appunto, i risultati. Al programmatore interessa la logica dell'elaborazione, cioè il codice Java, mentre al progettista grafico interessa la presentazione, cioè i marcatori HTML. Se i due aspetti progettuali sono mischiati, il programmatore e il progettista grafico devono

stare attenti a non interfe-rire nel lavoro altrui; cosa ancora peggiore, potrebbero entrambi provocare, inavverti-tamente, errori, facendo confusione con il codice o con i marcatori.

Un JavaBean è una classe con un

Quindi, gli inventori di JavaServer Pages raccomandano che qualsi-costruttore predefinito che met-

asi elaborazione non banale venga svolta in una classe Java separata, te a disposizione le proprie ca-usando uno dei meccanismi di JSP che rendono semplice tutto questo: ratteristiche mediante metodi get

la connessione a una pagina JSP di uno o più JavaBean. Praticamente, e set.

qualsiasi classe Java può essere un JavaBean, soddisfacendo un unico requisito tecnico: un JavaBean deve avere un costruttore pubblico senza parametri.

Il concetto più importante è che un JavaBean è una classe che mette a disposizione delle proprietà, alle quali si accede, eventualmente modificandole, tramite metodi che seguono una particolare convenzione sui nomi. Se il nome della proprietà è nomePro-prietà, e il suo tipo è Tipo, allora i metodi accessori e modificatori devono avere la forma seguente:

Tipo getNomeProprietà()

void setNomeProprietà(Tipo nuovoValore)

Ad esempio, se una proprietà di un JavaBean si chiama lastName ed è di tipo String, allora i metodi accessori e modificatori di tale proprietà sono: String getLastName()

void setLastName(String newValue)

Ovviamente, questa è la stessa convenzione che abbiamo usato per la maggior parte dei metodi accessori e modificatori.

Esiste un'eccezione alla convenzione sui nomi. Una proprietà booleana deve avere metodi accessori e modificatori aventi la forma seguente: boolean isNomeProprietà()

void setNomeProprietà(boolean nuovoValore)

JavaServer Pages

5

Ad esempio, la proprietà booleana married ha i metodi: boolean isMarried()

void setMarried(boolean newValue)

Infine, molti programmatori seguono un'ulteriore convenzione, che prevede che il nome della classe di un JavaBean termini in Bean, ma questo non è richiesto.

Ecco un semplice esempio di classe JavaBean:

```
public class PersonBean
{
// costruttore predefinito necessario
public PersonBean() { ... }
// proprietà lastName
public String getLastName() { ... }
public void setLastName(String newValue) { ... }
// proprietà married
public boolean isMarried() { ... }
public void setMarried(boolean newValue) { ... }
// variabili istanza
...
}
```

Questa classe ha due proprietà: lastName e married. Notate che il nome di una proprietà inizia con una lettera minuscola (come lastName), ma i metodi corrispondenti hanno una lettera maiuscola (getLastName). L'unica eccezione è che i nomi di proprietà possono essere tutti maiuscoli, come ID o URL, con i metodi corrispondenti getID o setURL.

In questo esempio, le due proprietà si possono leggere e scrivere, ma potete anche avere proprietà a sola lettura (con il solo metodo get o is) o a sola scrittura (con il solo metodo set).

Non dovreste fare alcuna ipotesi sulla rappresentazione interna delle proprietà della classe, che potrebbe avere

semplicemente una variabile istanza per ciascuna proprietà: private String lastName;

private boolean married;

Oppure, può darsi che la classe memorizzi lo stato di un suo esemplare in una base di dati, in modo che i metodi get e set, in realtà, contengano operazioni che coinvolgono una base di dati.

Cos'hanno di così speciale i JavaBean? Alcuni ambienti di programmazione, tra i quali JSP, forniscono comode modalità per accedere alle proprietà di un JavaBean senza dover invocare alcun metodo Java.

Una pagina JSP vi consente di

Per usare un JavaBean in una pagina JSP, usate la direttiva accedere alle proprietà di un Ja-jsp:useBean, fornendo il nome di un esemplare di JavaBean e il nome vaBean senza scrivere codice

della sua classe.

Java.

6

JavaServer Pages

Ad esempio

<jsp:useBean id="user" class="PersonBean"/> Questa direttiva invoca il costruttore predefinito della classe PersonBean per creare un oggetto di nome user. Ricordate che ogni JavaBean deve avere un costruttore predefinito, senza parametri.

Per impostare una proprietà, usate la direttiva setProperty:

<jsp:setProperty name="user" property="married" value="true"/> La direttiva jsp:setProperty costruisce il nome del metodo modificatore (setMarried), converte la stringa "true" nel valore booleano true, e invoca il metodo dell'oggetto user.

Al contrario, la direttiva jsp:getProperty invoca il metodo che legge una proprietà e converte in una stringa il valore letto. Tale stringa diviene parte della pagina HTML

prodotta dalla pagina JSP.

<jsp:getProperty name="user" property="married"/> Notate che il nome dell'oggetto JavaBean viene specificato dall'attributo id soltanto nella direttiva jsp:useBean

<jsp:useBean id="user" .../>

mentre le direttive jsp:setProperty e jsp:getProperty usano l'attributo name

<jsp:setProperty name="user" .../>

Quando usate i JavaBean, la pagina JSP contiene soltanto le relative direttive, senza codice Java; ovviamente, il codice Java è, però, contenuto nell'implementazione delle classi dei JavaBean. In questo modo, usando i JavaBean nelle pagine JSP, il codice viene separato dalla presentazione.

Ora mettiamo all'opera questi concetti in una vera pagina JSP. Supponiamo di voler visualizzare soltanto l'ora attuale, non l'intera data. Purtroppo, non c'è una classe Time che fornisca solamente l'ora, ma dovete comporre la stringa che vi interessa con un compositore che ignori i dati relativi a giorno, mese e anno, usando soltanto l'orario. Per ottenere un tale compositore, invocate il metodo statico getTimeInstance della classe DateFormat, una classe simile alla classe NumberFormat vista nel Capitolo 3.

DateFormat timeFormatter = DateFormat.getTimeInstance(); L'oggetto compositore risultante converte un oggetto di tipo Date in una stringa che contiene soltanto l'orario:

Date now = new Date(); // imposta now all'ora e alla data attuali String timeString = timeFormatter.format(now);

// es: "7:35:21 AM"

JavaServer Pages

7

Inseriamo questo codice nella classe di un JavaBean: ***File TimeFormatterBean.java***

import java.text.DateFormat;

import java.util.Date;

*/**

Questo JavaBean compone un orario a partire da una data completa.

```
*/
public class TimeFormatterBean
{
/**
Costruisce il compositore.
*/
public TimeFormatterBean()
{
timeFormatter = DateFormat.getTimeInstance();
}
/**
Proprietà di sola scrittura che contiene la data completa.
@param aDate la data da cui estrarre l'orario
*/
public void setDate(Date aDate)
{
theDate = aDate;
}
/**
Proprietà di sola lettura che contiene l'orario.
@return l'orario composto
*/
public String getTime()
{
String timeString = timeFormatter.format(theDate); return timeString;
}
private DateFormat timeFormatter;
private Date theDate;
}
```

Per usare questo JavaBean, dovete prima impostare la proprietà date con l'oggetto di tipo Date di cui volete estrarre l'orario, poi leggete la proprietà time per ottenere la stringa contenente l'orario estratto.

Ecco la pagina JSP che invoca il JavaBean per comporre l'orario.

File time.jsp

```
<jsp:useBean id="formatter" class="TimeFormatterBean"/>
<jsp:setProperty name="formatter" property="date"
value="<% new java.util.Date() %>"/>
```



```
<html>
<head>
<title>Date JSP</title>
</head>
<body>
<h1>Date JSP</h1>
<p>The current time is:
<jsp:getProperty name="formatter"
property="time"/>
</p>
</body>
</html>
```

Le prime due righe costruiscono il JavaBean compositore e impostano la sua proprietà usando la data attuale. In generale, è buona norma inserire le direttive per costruire i JavaBean e per impostarne le proprietà all'inizio del file JSP, prima dei contenuti HTML.

All'interno del corpo del documento HTML, la direttiva jsp:getProperty inserisce l'orario nel formato desiderato.

Per provare questa pagina JSP, dovere caricare nel server Web sia il file time.jsp, sia la classe TimeFormatterBean.class, inserendoli nelle seguenti cartelle:

- *il file time.jsp in c:\jakarta-tomcat\webapps\bigjava*
- *il file TimeFormatterBean.class in c:\jakarta-tomcat\webapps\bigjava\WEB-INF\classes*

Usate i JavaBean per separare la

Questo esempio mostra la separazione tra la presentazione e il codice presentazione HTML dai calcoli

Java. Un progettista grafico può migliorare la visualizzazione della pa-in Java.

gina modificando soltanto il documento HTML, relativamente al posizionamento delle informazioni, aggiungendo immagini, e così via, senza dover fare alcuna attività di programmazione. Viceversa, un programmatore può modificare il codice della classe del JavaBean senza fare alcuna progettazione Web.

Onestamente, la classe del JavaBean del nostro esempio è molto semplice, ma la miglioreremo nei paragrafi seguenti, aggiungendovi funzionalità significative. Questo modo di procedere, d'altronde, è tipico: molte applicazioni per il Web iniziano con una versione semplice, e nuove funzioni vengono aggiunte col tempo. Alcuni programmatori cedono alla tentazione e iniziano inserendo codice Java nelle loro pagine JSP, perché nei casi più semplici sembra che non valga la pena di realizzare classi separate per i JavaBean, ma, non appena l'applicazione diventa un po' più complessa, il codice finisce per diventare confuso. Se usate i JavaBean fin dall'inizio, anche nei casi più semplici, non avrete difficoltà a gestire un aumento di complessità.

Errori comuni 1

Sintassi JSP complicata

Fare errori con la sintassi JSP è molto facile, perché è un misto di HTML, marcatori specifici di JSP e codice Java. Inoltre, poiché il passo di compilazione JSP è nascosto al JavaServer Pages

9

programmatore, non si ottengono dei validi messaggi di errore. Ecco alcuni degli errori più comuni:

■ *Dimenticare il carattere / prima di una parentesi angolare di chiusura, come*

<jsp:useBean ... />. I marcatori JSP usano la sintassi XML e un carattere / è necessario (se non fornite un marcatore

</jsp:useBean> separato).

■ Usare lettere minuscole, come jsp:usebean invece di jsp:useBean. Il linguaggio HTML non è sensibile alla differenza tra maiuscole e minuscole, ma i marcatori JSP lo sono.

■ Dimenticare il carattere = dopo l'apertura <%. Esiste un diverso marcatore <% ...

%> che non inserisce il risultato nel file HTML che viene prodotto.

■ Confondere id e name. Nel marcatore useBean usate l'attributo id, mentre nei marcatori setProperty e getProperty usate l'attributo name.

■ Dimenticare le virgolette nei valori degli attributi. In HTML è consentito omettere queste virgolette, ma i marcatori JSP seguono la più rigida sintassi XML. Ad esempio, <jsp:useBean id=formatter ... /> è un errore. Usate, invece, id="formatter".

■ Inserire virgolette in stringhe racchiuse tra virgolette. Quando inserite un'espressione Java come attributo value di un marcatore JSP per i JavaBean, dovete rac-chiuderla tra virgolette, in questo modo:

<jsp:setProperty name="formatter" property="date"
value="<%= new java.util.Date() %>" />

Se l'espressione stessa contiene delle virgolette, le dovete tradurre in caratteri di escape, usando la barra rovesciata:

<jsp:setProperty name="formatter" property="zone"
value="<%= new java.util.TimeZone(\"Asia/Tokio\") %>" /> Oppure, potete usare apici singoli per racchiudere i valori degli attributi:

<jsp:setProperty name='formatter' property='zone'
value='<%= new java.util.TimeZone("Asia/Tokio") %>' /> Suggerimenti per la qualità 1

Rendete minimo l'uso di codice Java
in pagine JSP

Separare la presentazione dall'elaborazione è sempre una buona idea: in questo modo, gli esperti di interfaccia utente possono perfezionare la presentazione e i programmatori possono ottimizzare l'elaborazione.

Potete contraddire questo principio inserendo molto codice Java all'interno di marcatori JSP. Ad esempio, supponiamo che vogliate scrivere una pagina Web che genera un insieme di numeri della lotteria.

Ecco un esempio che non usa i JavaBean:

```
<html>
<head>
<title>Lottery JSP</title>
</head>
<body>
<h1>Lottery JSP</h1>
<p>Bet these numbers. They'll make you rich!
</p>
<ul>
<%
java.util.Random generator
= new java.util.Random();
int[] values = new int[49];
for (int i = 0; i < 49; i++)
values[i] = i + 1;
for (int i = 0; i < 6; i++)
{
int r = generator.nextInt(49 - i);
int c = values[r];
values[r] = values[48 - i];
out.print("<li>");
out.print(c);
out.print("</li>");
}
%>
```

```
</ul>
</body>
</html>
```

Siete in grado di capire quali modifiche occorre fare alla pagina HTML per aggiungere degli effetti grafici e altri abbellimenti? Siete in grado di capire cosa dovreste fare se voleste ordinare i numeri? Dato che la pagina mette insieme presentazione ed elaborazione, diventa molto difficile apportare qualsiasi cambiamento. È molto più sensato usare, invece, una classe LotteryBean:

```
<jsp:useBean id="lottery" class="LotteryBean"/>
<jsp:setProperty name="lottery" property="numbers"
value="49"/>
<jsp:setProperty name="lottery" property="choices"
value="6"/>
<html>
<head>
<title>Lottery JSP</title>
</head>
<body>
<h1>Lottery JSP</h1>
<p>Bet these numbers. They'll make you rich!
</p>
<jsp:getProperty name="lottery"
property="combination"/>
</body>
</html>
```


Una pagina JSP ben progettata usa i JavaBean per tutte le parti significative di codice Java, connettendoli fra loro con piccole porzioni di codice quando è necessario. Se vi accorgete che state inserendo in una pagina JSP dichiarazioni di variabili, diramazioni e cicli, allora è giunto il momento di usare un JavaBean.

C’è soltanto un’eccezione a questa regola: nel Paragrafo 6 vedrete pagine JSP che svolgono soltanto elaborazioni, seguite da una richiesta che rimanda a un’altra pagina. Poiché tali pagine non contengono alcun marcatore HTML, è accettabile che contengano molto codice Java.

Consigli pratici 1

Progettare un JavaBean

Un JavaBean è soltanto una normale classe Java, con due speciali caratteristiche.

- *Un JavaBean deve avere un costruttore predefinito.*
- *Dai marcatori JSP si possono invocare i loro metodi aventi la forma seguente Tipo getNomeProprietà()*

void setNomeProprietà(Tipo x)

Nel contesto della tecnologia JSP, l’uso più comune di un JavaBean consiste nella realizzazione di una struttura equivalente all’invocazione di un metodo. Tuttavia, poiché non si possono fornire parametri a un metodo di tipo get, occorre usare metodi di tipo set per impostare i valori di ingresso dell’elaborazione, per poi usare un metodo di tipo get per ottenerne il risultato. Quindi, l’invocazione di un metodo, come questa: result = metodo(param1, param2, ...)

diventa

```
<jsp:useBean id="aBean" class="MethodBean" />
<jsp:setProperty name="aBean" property="param1Name"
value="<%= param1 %>"/>
<jsp:setProperty name="aBean" property="param2Name"
value="<%= param2 %>"/>
...
<jsp:getProperty name="aBean" property="resultName"/>
```

Ecco le istruzioni per progettare la classe del JavaBean passo dopo passo.

Passo 1. *Decidete quale elaborazione ci si aspetta dal JavaBean.*

Quali sono i dati in ingresso? Qual è il risultato desiderato in uscita?

Passo 2. *Assegnate nomi di proprietà a ciascuna variabile di ingresso e di uscita.*

Passo 3. *Decidete il formato dell'uscita desiderata.*

Dovrebbe essere una stringa? Oppure un oggetto il cui metodo toString fornisce la stringa appropriata? Oppure, ancora, dovrebbe essere un frammento di codice HTML

per produrre un elenco puntato o una tabella?

Passo 4. *Realizzate il costruttore predefinito, senza argomenti.*

Tale costruttore inizializza tutte le variabili istanza che vengono riutilizzate ogni volta che viene eseguita l'elaborazione richiesta al JavaBean: ad esempio, impaginatori e compositori di stringhe, generatori di numeri casuali, e così via.

Passo 5. *Realizzate i metodi di tipo set per tutti i valori di ingresso dell'elaborazione.*

Questi metodi memorizzano semplicemente in variabili istanza i valori ricevuti.

Passo 6. *Fornite un metodo di tipo get che legga i parametri, elabori il risultato e lo restituisca.*

Poiché il risultato sarà inserito nella pagina HTML circostante, dovrebbe essere una stringa o un oggetto il cui metodo toString fornisca la stringa desiderata. La stringa risultante contiene spesso dei marcatori HTML.

Ecco un esempio. Supponete di voler generare una pagina Web che contiene un insieme di numeri della lotteria generati casualmente, come in Consigli per la qualità 1. Vogliamo simulare un'invocazione del tipo

result = combinations(49, 6)

Nel JavaBean che realizza l'estrazione della lotteria, abbiamo bisogno di due metodi di tipo set per impostare i numeri da cui estrarre e il numero di estrazioni da fare.

Chiamiamo numbers e choices queste proprietà; i metodi, quindi, sono: void setNumbers(int n)

void setChoices(int n)

L'elaborazione viene iniziata dall'invocazione di un metodo di tipo get; chiamiamo combination la proprietà che viene richiesta, per cui il metodo diventa: String getCombination()

Tale metodo restituisce una stringa di questo tipo:

*
22
17
18
16
*

*
27
36 Ecco il codice per il JavaBean completo:*

```
public class LotteryBean
{
public LotteryBean()
```



```
{
generator = new Random();
}
public void setNumbers(int n)
{
numbers = n;
}
public void setChoices(int n)
{
choices = n;
}
public String getCombination()
{
StringBuffer buffer = new StringBuffer();
buffer.append("<ul>");
int[] values = new int[numbers];
for (int i = 0; i < numbers; i++)
values[i] = i + 1;
for (int i = 0; i < choices; i++)
{
```

```
int r = generator.nextInt(numbers – i);
int c = values[r];
values[r] = values[numbers – 1 – i];
buffer.append("<li>");
buffer.append("" + c);
buffer.append("</li>");
}
buffer.append("</ul>");
return buffer.toString();
}
private Random generator;
private int numbers;
private int choices;
}
```

A questo punto la pagina JSP è pulita e senza codice, ma si potrebbe argomentare che, invece, il JavaBean non è molto pulito, in quanto genera codice HTML. Cosa succederebbe se volessimo inserire la combinazione vincente della lotteria in una tabella invece che in un elenco puntato? L'obiezione è corretta. L'Esercizio P.5 mostra come si possono usare due JavaBean: uno che compie l'elaborazione, producendo un vettore di numeri, e un altro che li impagina, trasformando il vettore in un elenco puntato.

3

Gestire i parametri della richiesta

Il programma del paragrafo precedente non rappresentava un esempio tipico, perché non riceve dati in ingresso dall'utente. Inoltre, fornendo all'utente l'orario relativo alla ***14***

collocazione geografica del server, non è molto utile per utenti situati in diversi fusi orari.

In questo paragrafo miglioreremo questa pagina JSP in modo da poter ricevere dall'utente i dati della città in cui si trova, visualizzando l'orario a essa relativo.

La libreria Java contiene una comoda classe TimeZone, che conosce i fusi orari mondiali. Un fuso orario è identificato da una stringa, come "America/Los_Angeles" oppure "Asia/Tokio". Il metodo statico getAvailableIDs restituisce un array di stringhe contenente tutti tali identificativi:

String[] zoneIDs = TimeZone.getAvailableIDs();

Esistono parecchie centinaia di identificativi di fusi orari. (Stiamo usando i fusi orari, in questo esempio, perché la classe TimeZone ci fornisce un'interessante fonte di dati, molto popolata. Più avanti nel capitolo, vedrete come recuperare dati da una base di dati, ma naturalmente ciò è più complesso.)

Il metodo statico getTimeZone restituisce un oggetto di tipo TimeZone relativo a una stringa identificativa assegnata:

String zoneID = "America/Los_Angeles";

TimeZone zone = TimeZone.getTimeZone(zoneID);

Dopo aver così costruito un oggetto di tipo TimeZone, lo potete usare insieme a un oggetto di tipo DateFormat per ottenere una stringa relativa a quel fuso orario.

DateFormat formatter = DateFormat.getTimeInstance(); formatter.setTimeZone(zone);

Date now = new date();

// supponiamo che il server sia a New York e che là sia mezzogiorno System.out.println(formatter.format(now));

// visualizza 9:00:00 AM

Ovviamente, non possiamo pretendere che l'utente conosca le stringhe che identificano i fusi orari, come "America/Los_Angeles", per cui faremo l'ipotesi che l'utente fornisca semplicemente il nome della città, come Los Angeles. Scriveremo un JavaBean che verifichi se tale stringa appare alla fine di un valido identificativo di fuso orario, dopo aver tradotto gli spazi in caratteri di sottolineatura. In caso affermativo, la pagina JSP compone e visualizza l'orario attuale in quel fuso orario; altrimenti,

visualizza un messaggio dicendo che quel nome di città non è stato trovato.

Dobbiamo affrontare il problema di come l'utente possa fornire il dato in ingresso alla pagina JSP. Occorre progettare un modulo HTML (form), come potete vedere in Figura 3, contenente un campo di testo con il nome della città e un pulsante per inviare i dati del modulo alla pagina JSP.

Ecco il codice HTML per il form:

File timezone.html

```
<html>
<head>
<title>Time Zone Form</title>
</head>
```

Figura 3
Il modulo HTML per inviare il nome della città

```
<body>
<form action="timezone.jsp" method="POST">
<p>City:
```

<input type=”text” name=”city”/>
<input type=”submit” value=”Get Time”/>
</p>
</form>
</body>
</html>

La Figura 3 mostra il modulo, dopo che l’utente ha inserito il nome di Quando un browser invia un form

a un server Web, invia i nomi e i

una città.

valori di tutti gli elementi del form.

Nel Paragrafo 4 vedrete in dettaglio come creare un modulo HTML; per ora, daremo solo un’occhiata più attenta a questo modulo semplice.

Per prima cosa notate l’attributo value del pulsante di invio: come potete vedere nella Figura 3, tale attributo specifica l’etichetta del pulsante, “Get Time”. Il campo di testo ha un attributo name, che non viene visualizzato sullo schermo, ma, quando il modulo viene inviato al server, ciò avviene sotto forma di una coppia di stringhe: il nome del campo di testo (in questo caso, city) e il suo contenuto (il nome di città inserito dall’utente).

L’attributo action dell’elemento form specifica lo URL del programma eseguito sul lato server che elabora i dati del form: nel nostro caso, vogliamo che In una pagina JSP potete accedere ai dati del form attraverso

i dati forniti dall’utente vengano letti dalla pagina timezone.jsp.

l’oggetto predefinito request.

Una pagina JSP può recuperare i dati digitati dall’utente nei campi di ingresso di un form interrogando l’oggetto speciale request, che è definito in ogni documento JSP come istanza della classe ServletRequest. (Come ve-16

JavaServer Pages

drete più avanti in questo capitolo, i servlet e le pagine JSP sono strettamente correlati.) Il metodo getParameter

restituisce il valore dell'elemento del form avente un certo nome. Ad esempio:

<%= request.getParameter("city") %>

fornisce il nome di città introdotto dall'utente.

Per ottenere l'orario in un particolare fuso, definirete una classe TimeZoneBean con una proprietà city. Ci sono due modi per impostare tale proprietà in modo che assuma il nome della città richiesta dall'utente. Ovviamente, potete inserire semplicemente l'invocazione del metodo getParameter all'interno della direttiva jsp:setProperty:

<jsp:setProperty name="zone" property="city"

value="<%= request.getParameter(\"city\") %>"/> Per fornire a un JavaBean un

Notate che le virgolette della stringa "city" devono essere precedute da parametro proveniente dalla ri-una barra rovesciata, perché sono contenute all'interno delle virgolette chiesta, esiste una comoda ab-dell'attributo value.

breviazione.

Ma esiste una comoda abbreviazione:

<jsp:setProperty name="zone" property="city"

param="city"/>

Quando usate l'attributo param in una direttiva jsp:setProperty, il valore della proprietà assume il valore del parametro proveniente dalla richiesta avente il nome specificato: questa abbreviazione rende molto semplice l'inizializzazione dei JavaBean con i parametri provenienti dalla richiesta.

Ecco il file JSP e la classe del JavaBean per gestire il fuso orario. La Figura 4 mostra ciò che viene prodotto da questo JSP.

File timezone.jsp

<jsp:useBean id="zone" class="TimeZoneBean"/>

<jsp:setProperty name="zone" property="date"

value="<% new java.util.Date() %>"/>

<jsp:setProperty name="zone" property="city"

```
param="city"/>
<html>
<head>
<title>Time Zone JSP</title>
</head>
<body>
<h1>Time Zone JSP</h1>
<p>The current time in
<%= request.getParameter("city") %> is:
<jsp:getProperty name="zone" property="time"/>
</p>
</body>
</html>
```

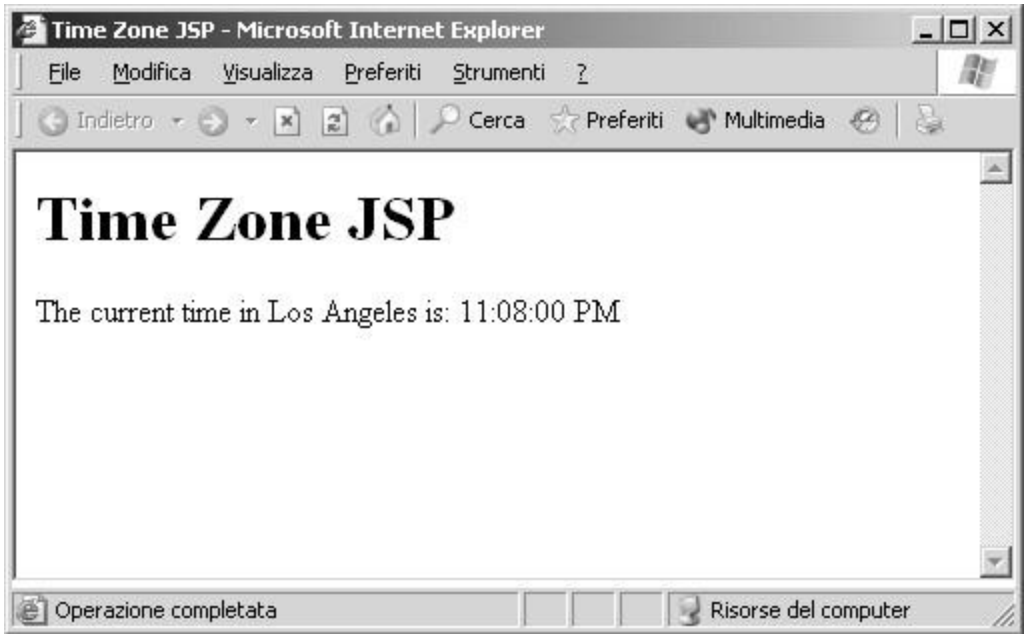


Figura 4
Il prodotto della pagina JSP per i fusi orari

File TimeZoneBean.java

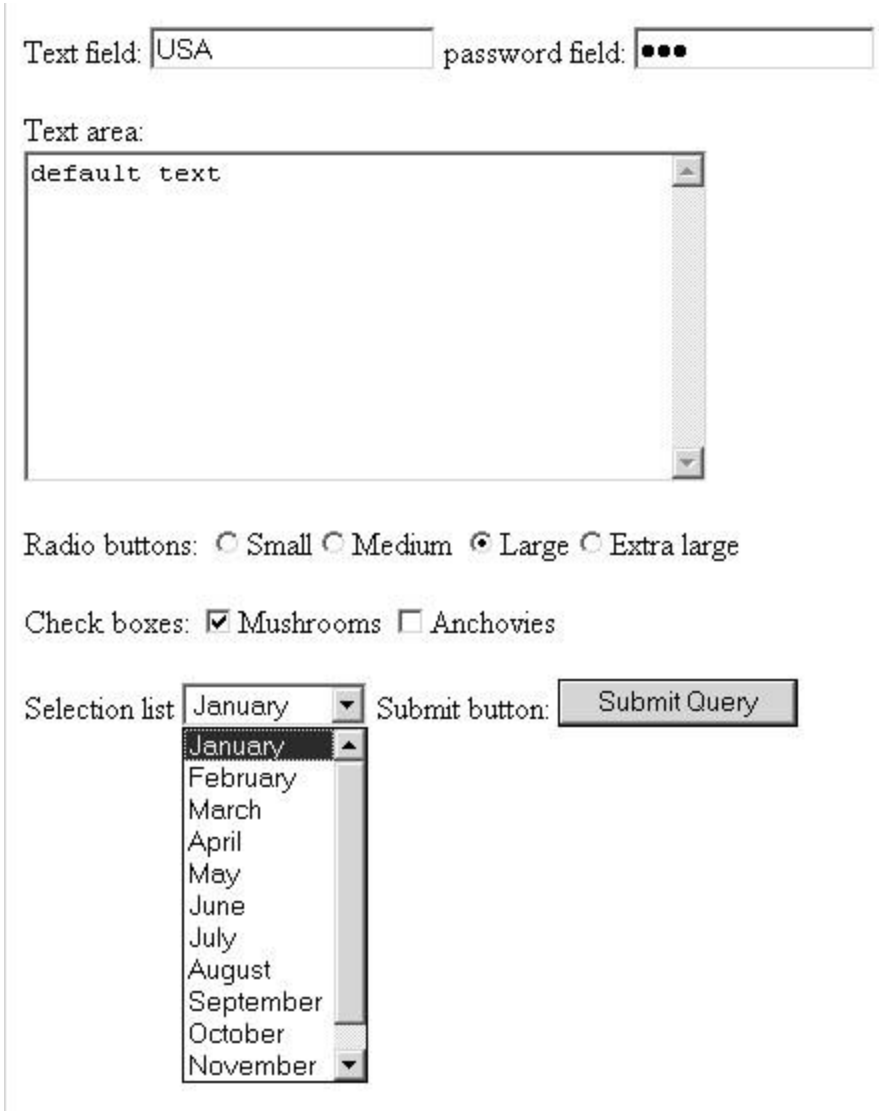
```
import java.text.DateFormat;
import java.util.Date;
```

import java.util.TimeZone;
*/**
Questo JavaBean compone in una stringa l'orario corrispondente a un certo istante in una certa città.
*/
public class TimeZoneBean
{
*/**
Inizializza il compositore di stringhe.
*/
public TimeZoneBean()
{
timeFormatter = DateFormat.getTimeInstance();
}
*/**
Proprietà di sola scrittura, "date".
@param aDate l'orario da utilizzare
*/
public void setDate(Date aDate)
{
theDate = aDate;
}
18
JavaServer Pages
*/**
Proprietà di sola scrittura, "city".
@param aCity la città per la quale si richiede l'orario
*/
public void setCity(String aCity)
{
city = aCity;
}
*/**
Proprietà di sola lettura, "available".
@return true se l'informazione del fuso orario è disponibile per questa città

```
    */
    public boolean isAvailable()
    {
    return getTimeZone(city) != null;
    }
    /**
    Proprietà di sola lettura, “time”.
    @return l’orario composto come stringa
    */
    public String getTime()
    {
    TimeZone zone = getTimeZone(city);
    if (zone == null) return “not available”;
    timeFormatter.setTimeZone(zone);
    String timeString = timeFormatter.format(theDate);
return timeString;
    }
    /**
    Cerca il fuso orario di una città.
    @param aCity la città per cui si cerca il fuso orario
    @return il fuso orario, oppure null se non lo si è trovato
    */
    private static TimeZone getTimeZone(String city)
    {
    String[] ids = TimeZone.getAvailableIDs();
    for (int i = 0; i < ids.length; i++)
    if (timeZoneIDmatch(ids[i], city))
    return TimeZone.getTimeZone(ids[i]);
    return null;
    }
    /**
    Verifica se un fuso orario corrisponde a una città.
    @param id la stringa che identifica il fuso orario (ad
esempio, “America/Los_Angeles”)
    @param aCity la città per cui si cerca il fuso orario (ad
esempio, “Los Angeles”)
```

@return true se c'è corrispondenza
**/*
JavaServer Pages
19
private static boolean timeZoneIDmatch(String id, String city)
{
String idCity = id.substring(id.indexOf('/') + 1); return idCity.replace('_', ' ').equals(city);
}
private DateFormat timeFormatter;
private Date theDate;
private String city;
}
Errori comuni 2
Mancata corrispondenza tra i nomi
dei parametri e quelli del form
È importante che il nome del campo di testo o del pulsante presente nel form sia uguale al nome del parametro utilizzato per interrogare la richiesta. Se i nomi non corrispondono, l'invocazione di getParameter restituisce null, oppure, ancor peggio, i contenuti di un diverso elemento del form.
Se modificate i nomi del form, dovete modificare anche il codice JSP. Ad esempio, se il form diventa:
<input type="text" name="cityname"/>
dovete modificare anche il codice nella pagina JSP, in questo modo:
<jsp:setProperty name="zone" property="city"
param="cityname"/>
Suggerimenti per la qualità 2
Usate URL relativi
Osservate l'attributo action nei marcatori del form presente nei file timezone.html e timezone.jsp.
<form action="timezone.jsp" method="POST"> Usate sempre uno URL relativo, non assoluto come questo:

<form
action="http://localhost:8080/bigjava/timezone.jsp"
method="POST">



Se usate soltanto URL relativi, è più facile spostare l'applicazione Web in un'altra collocazione.

4

Moduli (form) HTML

Un form HTML contiene elementi

Nel paragrafo precedente avete visto come una pagina JSP possa legge-dell'interfaccia utente, come cam-

re informazioni da un modulo (form) HTML: l'utente riempie il modulo pi di testo e pulsanti.

e il browser invia al server Web un insieme di coppie, formate dai nomi dei campi e dei relativi valori.

In questo paragrafo vedrete i tipi di campi che si possono inserire in un form HTML, che potete osservare nella Figura 5.

- *Campi di testo*
- *Campi per password*
- *Aree di testo*
- *Pulsanti radio*
- *Caselle di selezione o di spunta*
- *Elenchi per selezione*
- *Pulsanti di invio*
- *Campi nascosti*

Figura 5

Gli elementi
di un form HTML

La maggior parte degli elementi dei form ha la seguente sintassi:

<input type=tipo nome altriAttributi/>

In ogni elemento sarà presente un attributo name (con una sola eccezione: un pulsante di invio che sia l'unico presente in un form, non necessita di un nome) e potete anche aggiungere un attributo value, con un valore preimpostato. Ad esempio:

<input type="text" name="country" value="USA"/>

Avete già visto i campi di testo nell'esempio del fuso orario. Oltre al nome e a un possibile valore preimpostato, ci sono altri due attributi utili: l'attributo size indica il numero di caratteri che vengono visualizzati nel campo e l'attributo maxlength limita il numero di caratteri che possono essere digitati dall'utente.

Il tipo password è simile al tipo text, ma tutti i caratteri digitati dall'utente vengono visualizzati come un unico carattere speciale, solitamente un asterisco o un palli-no: usate tali campi per informazioni riservate che non devono essere sbirciate stando dietro le spalle dell'utente, come le parole d'accesso e i numeri di conto bancario. Ad esempio:

<input type="password" name="accountnumber" size="16"

maxlength="16"/>

Attributo

Usato negli elementi

name

Tutti gli elementi

value

Tutti gli elementi per inserimento dati

size

text, password

maxlength

text, password

rows

textarea

cols

textarea

checked

radio, checkbox

Per consentire all'utente di inserire dati in ingresso che si estendano per più righe, usate un'area di testo. Le aree di testo non usano un elemento di tipo input, ma un elemento di tipo textarea, che racchiude un eventuale valore preimpostato:

<textarea name="..." rows="..." cols="..."> testo preimpostato

</textarea>

Gli attributi rows e cols specificano il numero di righe e di colonne che vengono visualizzate, ma l'utente può anche digitare più testo di quanto ne possa essere contenuto:

l’area di testo scorrerà per accogliere testo in qualsiasi quantità.

I pulsanti radio hanno la forma seguente:

<input type=”radio”

name=nome del gruppo di pulsanti

value=valore della selezione

ulteriori attributi/>

I pulsanti radio lavorano in gruppi: se l’utente seleziona un pulsante di un gruppo, il browser deseleziona automaticamente il pulsante selezionato in precedenza. Il form HTML dovrebbe specificare quale elemento deve apparire come selezionato nel momento in cui il form viene visualizzato per la prima volta, fornendo un attributo checked=”checked”. Ecco un esempio:

<input type=”radio” name=”shirtsize”

value=”S”/>Small

<input type=”radio” name=”shirtsize”

value=”M”/>Medium

<input type=”radio” name=”shirtsize”

value=”L” checked=”checked”/>Large

<input type=”radio” name=”shirtsize”

value=”XL”/>Extra large

Per sapere quale sia la disposizione visiva risultante per questo gruppo di pulsanti, osservate la Figura 5. Notate che l’ etichetta del pulsante non fa parte dell’elemento di tipo input, ma viene specificata con una normale stringa di testo HTML oppure con un’immagine. L’attributo value indica il valore che deve essere inviato dal browser al server quando viene inviato il form: ad esempio, se l’utente seleziona il pulsante che si trova accanto all’etichetta “Large”, il browser invia al server la coppia nome/valore shirtsize=L (e non shirtsize=Large).

Gli elementi che rappresentano caselle di selezione o di spunta si codificano in modo simile ai pulsanti radio; ad

esempio:

<input type=”checkbox” name=”topping” value=”mushrooms” checked=”checked”/>Mushrooms

<input type=”checkbox” name=”topping” value=”anchovies”/>Anchovies

In un gruppo di caselle di spunta possono esserci più caselle selezionate contemporaneamente e, anche, più caselle con l’attributo checked impostato al valore “checked”.

Se entrambe le caselle sono selezionate, il browser invia al server la stringa seguente: topping=mushrooms&topping=anchovies

Se volere fornire all’utente in modo compatto un elevato numero di opzioni tra cui sceglierne una, usate un elenco per selezione, che è il secondo tipo di elementi in un form che non usa l’elemento di tipo input, ma si inseriscono elementi di tipo option in un elemento di tipo select, come in questo esempio:

```
<select>
<option value=”1” selected=”selected”>January</option>
<option value=”2”>February</option>
<option value=”3”>March</option>
<option value=”4”>April</option>
<option value=”5”>May</option>
<option value=”6”>June</option>
```



```
<option value=”7”>July</option>
<option value=”8”>August</option>
<option value=”9”>September</option>
<option value=”10”>October</option>
<option value=”11”>November</option>
<option value=”12”>December</option>
</select>
```

Ciascuna opzione ha un valore da inviare al server e contiene il testo a essa associato, che viene visualizzato in un menu a discesa.

Nella sua forma più elementare, un pulsante di invio è così semplicemente codifi-cato:

<input type="submit"/>

Questo marcatore visualizza un pulsante di invio predefinito (osservate la Figura 5), di cui potete modificare l'etichetta specificandone l'attributo value:

<input type="submit" value="Get time"/>

Potete anche inserire, in una stessa pagina, più pulsanti di invio: in questo caso, dovreste usare gli attributi name e value, in modo che la pagina JSP possa sapere quale pulsante è stato premuto. Ad esempio:

<input type="submit" name="show" value="Next"/>

<input type="submit" name="show" value="Previous"/> Infine, potete inserire in un form dei campi nascosti: il browser non li visualizza, ma invia al server le coppie nome/valore corrispondenti, quando l'utente invia il form.

Alcune applicazioni Web usano i campi nascosti per conservare informazioni che pro-vengono da un'interazione precedente, ad esempio per fornire a un programma che elabora il form attuale il numero di conto corrente selezionato dall'utente in un form precedente. La sintassi è banale:

<input type="hidden" name="accountnumber"

value="YVV-11534-449-6"/>

Un form HTML deve specificare

Tuttavia, questa tecnica è inusuale nelle pagine JSP: come vedrete nel lo URL del programma che, sul

prossimo paragrafo, le pagine JSP sono in grado di tenere traccia delle server, elabora i dati del form

sessioni, per memorizzare in modo appropriato e comodo informazioni stesso.

che debbano essere disponibili a più form. Infine, inserite in un elemento di tipo form tutti gli elementi che appartengono a un form.

<form action="timezone.jsp" method="POST">

...

</form>

L'attributo action del marcatore form contiene lo URL della pagina JSP che elabora i dati del form stesso. L'attributo method dovrebbe avere il valore POST. (Ci sono due metodi per inviare dati a un server Web: il metodo GET e il metodo POST. Il metodo POST è più adatto per i dati di un form, perché non ha alcun limite sulla quantità di dati.)

24

JavaServer Pages

5

Gestione delle sessioni

Una sessione è una sequenza di

Considerate una tipica applicazione di commercio elettronico sul Web.

pagine richieste da uno stesso

Quando aggiungete articoli al vostro carrello degli acquisti, il carrello browser a uno stesso server Web.

ricorda ciò che aggiungete. Per realizzare un comportamento di questo tipo in una applicazione Web, dovete gestire le sessioni degli utenti. Una sessione è una sequenza di pagine richieste da uno stesso browser a uno stesso server Web.

Per gestire una sessione in una

Con la tecnologia JSP, la gestione delle sessioni è molto semplice. Se sequenza di pagine JSP, usate un

contrassegnate un JavaBean con l'attributo scope="session", lo stesso JavaBean con visibilità di ses-oggetto viene usato per tutta la sessione. Ad esempio, la direttiva se-sione.

guente dichiara un oggetto che rappresenta un carrello degli acquisti che è disponibile per un'intera sessione di un

utente.

<jsp:useBean id=”cart” class=”ShoppingCartBean”
scope=”session”/>

Quando l’utente richiede per la prima volta la pagina JSP che contiene questa direttiva, viene costruito un nuovo oggetto di nome cart. Quando l’utente torna alla stessa pagina Web oppure visita un’altra pagina che fa parte della stessa applicazione Web, l’oggetto di nome cart è ancora attivo. Di conseguenza, quando vengono aggiunti articoli al carrello, tali articoli si trovano ancora al loro posto nel carrello quando l’utente torna a una pagina che fa parte di questa stessa applicazione. Viene, invece, creato un diverso carrello per ciascun utente.

Al contrario, se una direttiva per i JavaBean non contiene l’attributo scope, l’esemplare di JavaBean ha visibilità di pagina: in tal caso, viene creato un nuovo oggetto ogni volta che la pagina viene visitata.

Usate la visibilità di sessione per tutte quelle informazioni che devono essere conservate di pagina in pagina mentre l’utente utilizza la vostra applicazione Web; tipiche informazioni di questo tipo sono il nome dell’utente, gli articoli ordinati, le opzioni di personalizzazione, e così via.

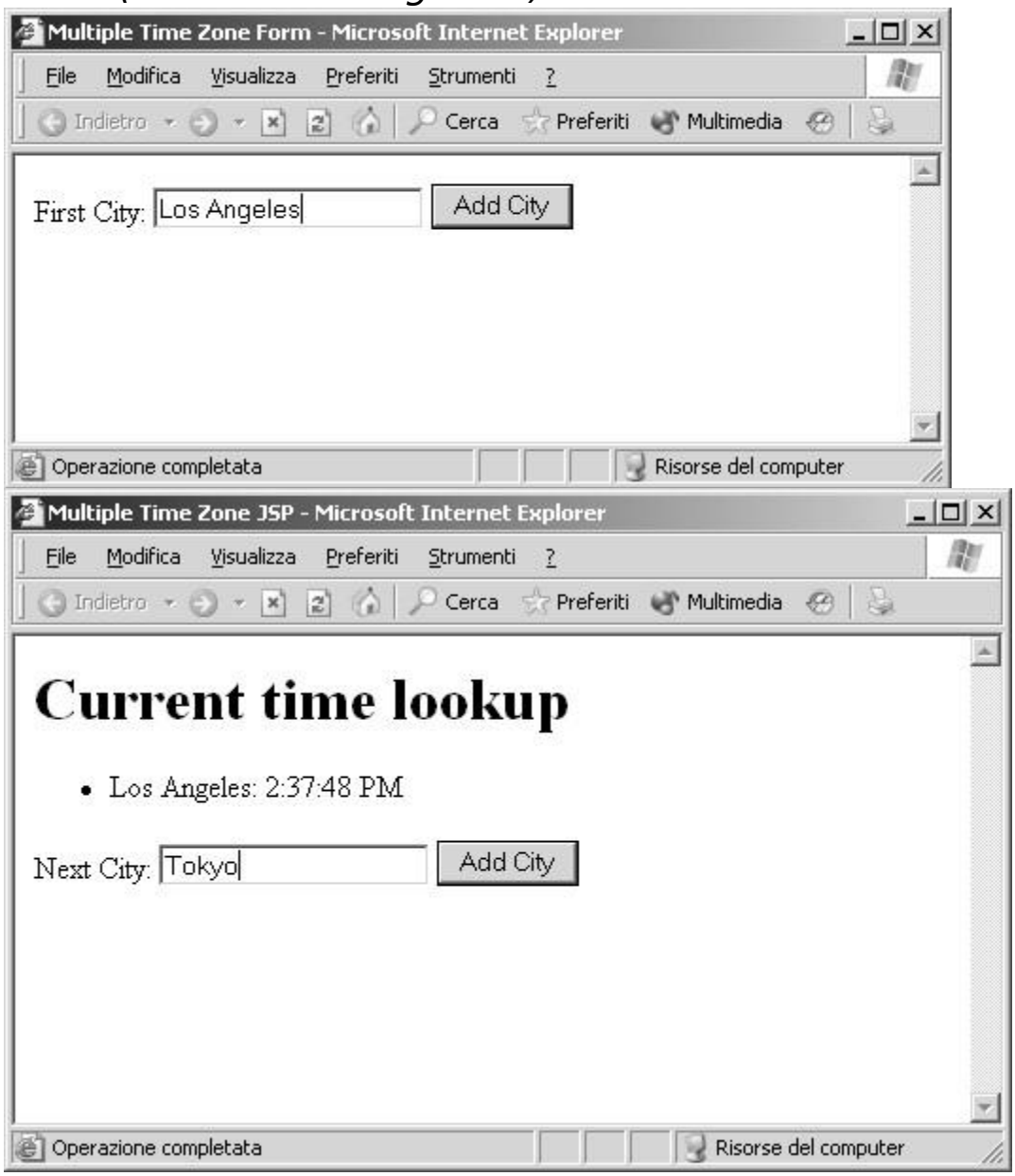
Scriviamo, ora, un semplice programma che mostri la visibilità di sessione all’opera. Invece di visualizzare l’orario di una singola località, consentiremo agli utenti di aggiungere ulteriori città, una per volta. Nel form HTML iniziale, questa applicazione Web chiede il nome della prima città (osservate la Figura 6).

La pagina JSP visualizza, quindi, la prima città, insieme a un form con il quale si può inserire la città successiva (osservate la Figura 7).

Potete aggiungere tutte le città che volete, e la pagina JSP visualizzerà un elenco di tutte le città e del loro orario attuale (osservate la Figura 8). Tutte le città vengono memorizzate in un oggetto di tipo MultiZoneBean. Il metodo setCity aggiunge una città a tale oggetto. Poiché l’oggetto

ha visibilità di sessione, il medesimo oggetto conserva un elenco di città di dimensione sempre crescente.

Come nel paragrafo precedente, usiamo un form HTML per l'introduzione iniziale dei dati da parte dell'utente. Tale form invia il nome di una città a un file JSP, che visualizza un elenco di città e i relativi orari locali. Il file JSP contiene, esso stesso, un form, che invia dati al file JSP medesimo. Ogni volta che l'utente invia il form, viene aggiunta un'altra città alla lista (osservate la Figura 9).



25

Figura 6

Richiesta della prima città di un elenco

Figura 7

Richiesta della città successiva

Conserviamo le città in un oggetto di tipo MultiZoneBean, che viene creato per ciascuna sessione:

<jsp:useBean id="zones" class="MultiZoneBean" scope="session"/>

Il suo metodo setCity aggiunge una città all'elenco delle città e viene invocato ogni volta che viene richiesta la pagina.

26

JavaServer Pages

Figura 8

Visualizzazione degli orari in più località

Figura 9

Struttura

dell'applicazione Web

per fusi orari multipli

La nuova città è il valore del campo di testo avente nome city:

<jsp:setProperty name="zones" property="city"

param="city"/>

La proprietà times dell'oggetto di tipo MultiZoneBean contiene il codice HTML di un elenco puntato con i nomi delle città e i relativi orari locali.

Potete verificare che MultiZoneBean ha visibilità di sessione facendo partire due browser separati (come, ad esempio, Netscape Navigator e Internet Explorer), facendo esaminare a entrambi il file multizone.html. Potete poi costruire due elenchi di città: ciascun elenco ricorda i vostri inserimenti precedenti, perché il server Web man-JavaServer Pages

tiene una sessione per ciascun browser connesso, ma gli elenchi di città sono indipen-denti l'uno dall'altro, essendo contenuti in due oggetti diversi, uno per ciascuna sessione.

File multizone.html

```
<html>
<head>
<title>Multiple Time Zone Form</title>
</head>
<body>
<form action="multizone.jsp" method="POST">
<p>First City:
<input type="text" name="city"/>
<input type="submit" value="Add City"/>
</p>
</form>
</body>
</html>
```

File multizone.jsp

```
<jsp:useBean id="zones" class="MultiZoneBean"
scope="session"/>
<jsp:setProperty name="zones" property="date"
value="<%= new java.util.Date() %>"/>
<jsp:setProperty name="zones" property="city"
param="city"/>
<html>
<head>
<title>Multiple Time Zone JSP</title>
</head>
<body>
<h1>Current time lookup</h1>
<ul>
<jsp:getProperty name="zones" property="times"/>
</ul>
<form action="multizone.jsp" method="POST">
<p>Next City:
<input type="text" name="city"/>
<input type="submit" value="Add City"/>
</p>
</form>
</body>
</html>
```

File MultiZoneBean.java

```
import java.text.DateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.TimeZone;
```



```
/**
Questo JavaBean impagina un elenco puntato di città e
dei relativi orari attuali.
*/
public class MultiZoneBean
```

```
{
/**
Inizializza il compositore degli orari e l'elenco di città.
*/
public MultiZoneBean()
{
timeFormatter = DateFormat.getTimeInstance();
cities = new ArrayList();
}
/**
Proprietà di sola scrittura, "date".
@param aDate l'orario da utilizzare
*/
public void setDate(Date aDate)
{
theDate = aDate;
}
/**
Proprietà di sola scrittura, "city".
@param aCity la città da aggiungere all'elenco di città
*/
public void setCity(String aCity)
{
cities.add(aCity);
}
/**
Proprietà di sola lettura, "times".
@return una stringa contenente il codice HTML per un
elenco puntato di città e dei rispettivi orari locali
*/
public String getTimes()
{
StringBuffer buffer = new StringBuffer();
for (int i = 0; i < cities.size(); i++)
{
String city = (String)cities.get(i);
```

```
buffer.append("<li>");
buffer.append(city);
buffer.append(": ");
TimeZone zone = getTimeZone(city);
if (zone == null)
buffer.append("not available");
else
{
```

```
timeFormatter.setTimeZone(zone);
String timeString
= timeFormatter.format(theDate);
buffer.append(timeString);
}
buffer.append("</li>");
}
return buffer.toString();
}
/**
Cerca il fuso orario di una città.
@param aCity la città per cui si cerca il fuso orario
@return il fuso orario, oppure null se non lo si è trovato
*/
private static TimeZone getTimeZone(String city)
{
String[] ids = TimeZone.getAvailableIDs();
for (int i = 0; i < ids.length; i++)
if (timeZoneIDmatch(ids[i], city))
return TimeZone.getTimeZone(ids[i]);
return null;
}
/**
Verifica se un fuso orario corrisponde a una città.
@param id la stringa che identifica il fuso orario (ad esempio, "America/Los_Angeles")
```

@param aCity la città per cui si cerca il fuso orario (ad esempio, “Los Angeles”)
@return true se c’è corrispondenza
**/*
private static boolean timeZoneIDmatch(String id, String city)
{
String idCity = id.substring(id.indexOf(‘/’) + 1); return idCity.replace(‘_’, ‘ ‘).equals(city);
}
private DateFormat timeFormatter;
private Date theDate;
private ArrayList cities;
}
Argomenti avanzati
Lo stato di una sessione
Come già sapete, il protocollo HTTP è privo di stato (stateless): un browser invia una richiesta a un server Web, il quale invia la risposta e si disconnette. Questo funzionamento è diverso da quello di altri protocolli, come POP, tramite il quale dapprima il ***30***
JavaServer Pages
client di posta elettronica si collega al server, eseguendo una procedura di login, e poi invia comandi per recuperare e cancellare i messaggi di posta elettronica. Il client di posta rimane collegato finché non chiude esplicitamente la connessione; al contrario, un browser effettua una nuova connessione al server Web per ogni pagina Web, e il server non ha modo di sapere che tali connessioni hanno origine dallo stesso browser.
I cookie (“biscottini”) sono stati inventati per superare questa limitazione. Un cookie è formato da una breve stringa che il server Web invia al browser, e che il browser rimanda a quel medesimo server insieme a tutte le successive richieste: in questo modo, il server può tenere assieme il flusso delle richieste. Può darsi che abbiate

sentito alcuni difensori della privacy protestare a proposito dei cookie, che, però, non sono intrinsecamente malvagi: quando vengono usati per stabilire una sessione o per ricordare le informazioni di connessione, possono rendere più facili da utilizzare le applicazioni Web, ma quando vengono usati per tenere traccia della vostra identità mentre navigate sul Web, allora ci possono essere delle violazioni della privacy.

Alcune persone disabilitano l'uso dei cookie: in tal caso, le applicazioni Web devono usare un diverso schema per identificare una sessione, solitamente inserendo un valore identificativo della sessione nello URL della richiesta, oppure un campo nascosto in un form. Fortunatamente, il meccanismo di funzionamento delle sessioni di JSP

è abbastanza astuto da usare automaticamente gli URL identificativi della sessione qualora un browser non gestisca i cookie.

6

Diramazioni e rimandi
ad altre pagine

Progettando un'applicazione Web con le pagine JSP, si usano i marcatori HTML per costruire l'interfaccia utente. Se alcune parti dell'interfaccia utente sono dinamiche, cioè dipendono dai dati che si stanno visualizzando all'utente, userete i JavaBean per generarle, ma, a volte, l' intera pagina dipende dai dati che vengono visualizzati. Ad esempio, considerate la frequente situazione della gestione degli errori: se l'utente fornisce valori errati in ingresso, vorrete mostrare una pagina Web che indica all'utente quale errore ha commesso e fornisce la possibilità di rimediare; d'altra parte, se i dati forniti in ingresso erano corretti, volete mostrare una pagina Web con le informazioni richieste dall'utente.

Una pagina JSP può inviare a

In tale situazione, potete usare la tecnica della "diramazione e ri-un'altra pagina la richiesta rice-

mando” (branch and forward), che presenteremo in questo paragrafo.

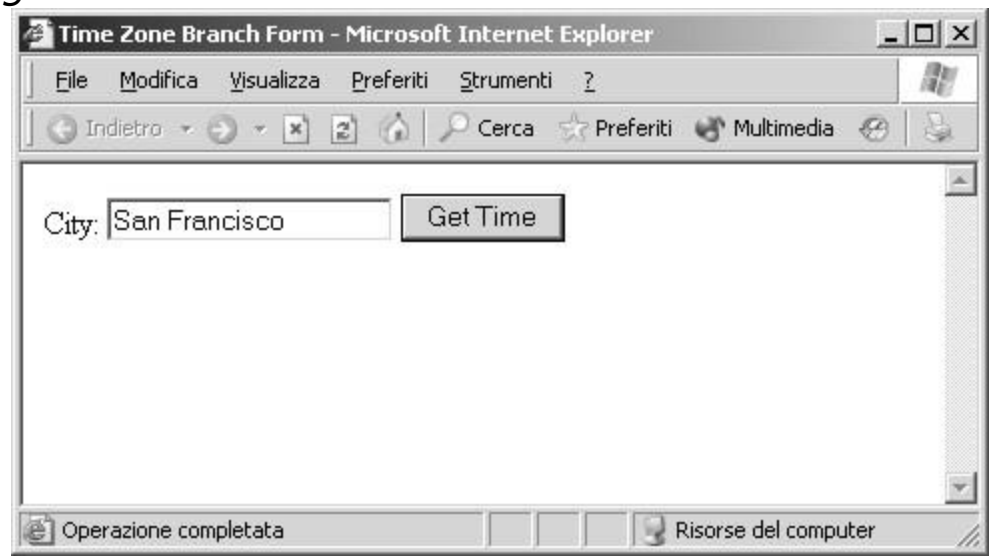
vuta, una tecnica utile per realiz-

Con questa tecnica, si invia la richiesta dell’utente a una pagina JSP che zare diramazioni.

verifica la correttezza dei dati forniti dall’utente, senza generare codice HTML, ma valutando una condizione e usando la direttiva jsp:forward per selezionare un’altra pagina JSP. La direttiva

<jsp:forward page=”url”/>

indica al motore JSP di caricare un’altra pagina al posto di quella attuale. I dati della richiesta relativi alla pagina attuale, e tutti i JavaBean con visibilità di tipo scope=”request”, vengono passati alla nuova pagina. La pagina che contiene il codice



di “diramazione e rimando” fornisce a un JavaBean i dati della richiesta e termina con una verifica come questa:

```
<%
if (condizione)
{
%>
<jsp:forward page="url1"/>
```

```
<%
}
else
{
%>
<jsp:forward page="url2"/>
<%
}
%>
```

Notate che gli enunciati Java sono racchiusi tra marcatori <%...%>, che sono diversi dai marcatori <%=...%> che avete visto per le espressioni semplici. Come abbiamo già detto, è lecito aggiungere enunciati Java a una pagina JSP, anche se ciò viola la separazione tra presentazione ed elaborazione. Tuttavia, in pagine che non contengono informazioni relative alla presentazione, è accettabile la presenza di codice Java. In questo caso, la presentazione si trova nelle pagine JSP che vengono caricate attraverso le direttive jsp:forward.

Nell'esempio che segue, usiamo questa tecnica per gestire i nomi delle città per le quali non è noto il fuso orario. Iniziamo con lo stesso form HTML di prima (osservate la Figura 10 e il file zonebranch.html), ma inviamo i dati del form alla pagina non visuale zonebranch.jsp.

Figura 10

Inizio della ricerca
del fuso orario

File zonebranch.html

```
<html>
<head>
<title>Time Zone Branch Form</title>
</head>
<body>
<form action="zonebranch.jsp" method="POST">
```

```
<p>City:
<input type="text" name="city"/>
<input type="submit" value="Get Time"/>
</p>
</form>
</body>
</html>
```

File zonebranch.jsp

```
<jsp:useBean id="zone" class="TimeZoneBean"
scope="request"/>
<jsp:setProperty name="zone" property="date"
value="<%= new java.util.Date() %>"/>
<jsp:setProperty name="zone" property="city"
param="city"/>
<%
if (zone.isAvailable())
{
%>
<jsp:forward page="zoneresult.jsp"/>
<%
}
else
{
%>
<jsp:forward page="zoneerror.jsp"/>
<%
}
%>
```

Un JavaBean con visibilità di tipo

La pagina zonebranch.jsp costruisce un oggetto di tipo TimeZoneBean e richiesta (request) rimane atti-ne imposta le proprietà date e city. Quindi, la logica di diramazione vo quando la richiesta viene in-invoca il metodo isAvailable della classe TimeZoneBean, per vedere se viata di rimando a un'altra pa-esistono informazioni relative al fuso orario della città in esame: se il gina.

metodo restituisce true, la richiesta viene inviata di rimando alla pagina zoneresult.jsp, che mostra il risultato della ricerca successiva (osservate la Figura 11).

Notate l'attributo scope relativo a TimeZoneBean nella pagina zonebranch.jsp:

<jsp:useBean id="zone" class="TimeZoneBean"

scope="request"/>

La visibilità di richiesta (request) indica che il JavaBean è attivo nella pagina attuale e in qualsiasi altra pagina a cui venga inviata, di rimando, la richiesta. Di conseguenza,

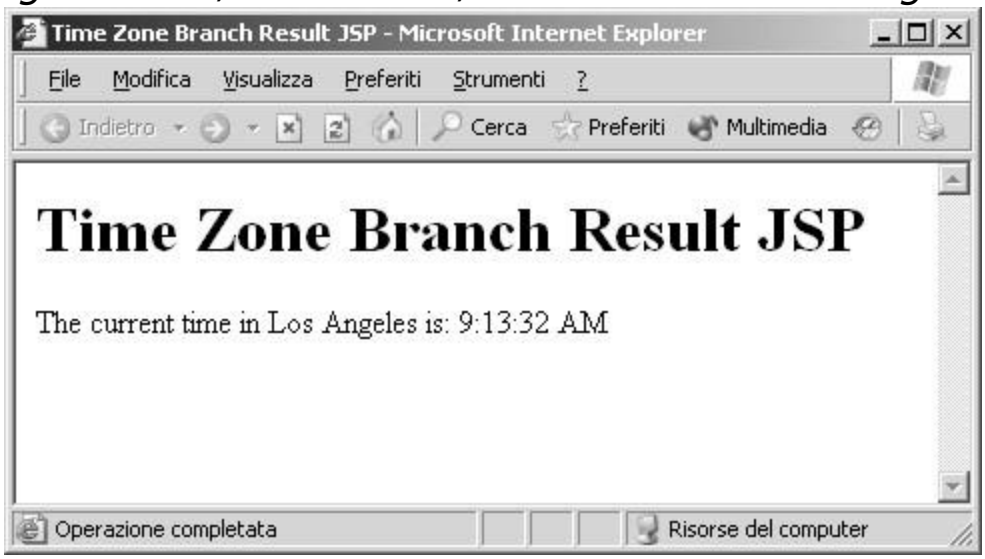


Figura 11

Una ricerca andata a buon fine

la pagina zoneresult.jsp può continuare ad accedere al JavaBean zone. Inoltre, anche l'oggetto originario di tipo request viene trasmesso alla nuova pagina. Notate come la pagina zoneresult.jsp sia in grado di leggere il valore del parametro city che era stato inviato dal form alla pagina zonebranch.jsp.

File zoneresult.jsp

<html>

<head>

```
<title>Time Zone Branch Result JSP</title>
</head>
<body>
<h1>Time Zone Branch Result JSP</h1>
<p>The current time in
<%= request.getParameter("city") %>
is:
<jsp:getProperty name="zone" property="time"/>
</p>
</body>
</html>
```

Infine, vediamo cosa succede se la ricerca è infruttuosa: in tal caso, la richiesta viene inviata alla pagina zoneerror.jsp (osservate la Figura 12), nella quale l'utente ha la possibilità di introdurre il nome di un'altra città. I dati del form vengono nuovamente inviati alla pagina zonebranch.jsp. La Figura 13 mostra la struttura di questa applicazione Web.

Figura 12
Una ricerca infruttuosa

Figura 13

Struttura
dell'applicazione
per la ricerca
di un fuso orario

File zoneerror.jsp

```
<html>
<head>
<title>Time Zone Error JSP</title>
</head>
<body>
<p>Unfortunately, no information is available for
<%= request.getParameter("city") %>.
Please enter another city.
</p>
<form action="zonebranch.jsp">
```

JavaServer Pages

35

```
<p>City:
<input type="text" name="city"/>
<input type="submit" value="Get Time"/>
</p>
</form>
</body>
</html>
```

Consigli pratici 2

Realizzare un'applicazione Web

Passo 1. *Su fogli di carta distinti, tracciate uno schizzo della schermata di ciascuna pagina Web dell'applicazione.*

Partite dalla schermata iniziale, che spesso gestisce l'autenticazione dell'utente (login). In ciascuna schermata saranno presenti uno o più pulsanti che inviano dati al server e provocano un cambio di pagina (oppure ripresentano la stessa pagina, con alcune informazioni aggiuntive). Assicuratevi di aver previsto le schermate di errore per segnalare dati d'ingresso non validi.

Passo 2. *Tracciate un diagramma delle transizioni, simile a quello di Figura 13.*

In ciascuna pagina, disegnate una freccia che punti alla pagina successiva ed etichet-tatela con il nome del pulsante. Se c'è bisogno di prendere una decisione (ad esempio, per verificare un potenziale errore), inserite una pagina JSP non visuale.

Passo 3. *In ciascuna pagina Web, contrassegnate le parti statiche e dinamiche.*

Le parti statiche rimangono sempre uguali (ad esempio, i titoli, le immagini che rappresentano marchi aziendali o istituzionali, le istruzioni, e così via), mentre le parti dinamiche cambiano in conseguenza dei dati forniti in ingresso dall'utente.

Passo 4. *Le parti dinamiche devono essere generate dai JavaBean.*

Progettate un insieme di JavaBean che generino le parti dinamiche. Identificate i metodi di tipo get e set. Spesso la stessa informazione si trova in pagine diverse: in questo caso, potete condividere gli stessi JavaBean.

Passo 5. *Controllate le decisioni che devono essere prese dalle pagine JSP non visuali (come, ad esempio, il controllo degli errori).*

Tali pagine JSP non visuali hanno bisogno di aiuto da parte dei JavaBean che avete già sviluppato? C'è bisogno di aggiungere metodi o nuovi JavaBean?

Passo 6. *Usando un editor HTML, disegnate le pagine Web di base.*

Se il vostro editor HTML vi consente l'inserimento di marcatori JSP, usateli; altrimenti aggiungeteli con un editor di testo. Per generare le parti dinamiche di una pagina, vi servono i marcatori jsp:getProperty. Nella prima pagina dell'applicazione, dovete creare gli esemplari di tutti i JavaBean che hanno visibilità di sessione, usando il

marcatore jsp:useBean (questa caratteristica è spesso quella più adatta per un JavaBean inserito in un'applicazione Web). All'inizio di ciascuna pagina, inserite i marcatori jsp:setProperty per impostare i parametri ricevuti nella richiesta e altre variabili.

Passo 7. *Implementate il codice Java dei JavaBean e compilateli.*

Passo 8. *Caricate la vostra applicazione nel server.*

Copiate i file JSP nella cartella dell'applicazione Web e i file con il bytecode nella sottocartella WEB-INF/classes.

7

Un'applicazione a tre livelli

Un'applicazione a tre livelli ha li-

In questo esempio conclusivo sulla tecnologia JSP, vedrete un'applica-velli distinti per la presentazio-

zione Web avente una struttura molto comune: useremo una base di ne, l'elaborazione e la memoriz-dati per memorizzare le informazioni e miglioreremo l'esempio dei fusi zazione dei dati.

orari memorizzando in tale base di dati città aggiuntive che non siano note alla classe TimeZone. Un'applicazione di questo tipo viene chiamata applicazione a tre livelli (three-tier o 3-tier), perché è composta da tre diversi livelli o strati di software:

- *Il livello di presentazione: il browser Web*
- *Il livello di elaborazione ("business logic"): il motore JSP, le pagine JSP e i JavaBean*
- *Il livello di memorizzazione: la base di dati*

Il livello intermedio (middle tier) viene spesso chiamato "business logic" perché, nelle applicazioni commerciali, contiene le regole che vengono usate per prendere le decisioni: quali prodotti offrire, quanto farli pagare, a chi concedere credito, e così via. Nei nostri semplici esempi, la logica dei fusi orari rappresenta tale livello: la parte "intelli-gente" dell'applicazione si trova, quindi, nello strato intermedio. I livelli di presentazione e memorizzazione

usano browser e sistemi di basi di dati già disponibili sul mercato, con form e tabelle specifiche per l'applicazione. Osservate la Figura 14 per avere un diagramma schematico dei livelli; si è soliti, anche, suddividere il livello intermedio in ulteriori livelli, giungendo così a un'applicazione n-tier.

Confrontate l'architettura a tre livelli con la più tradizionale architettura client-server o a due livelli che avete visto nel capitolo dedicato alla programmazione delle basi di dati. In tale architettura, uno dei livelli è il server della base di dati, a cui accedono più programmi client, in esecuzione su diversi calcolatori. Ciascun client ha uno strato di presentazione, solitamente con un'interfaccia utente programmata appositamente, e uno strato contenente la business logic (osservate la Figura 15). Notate che il JavaServer Pages

Figura 14

Architettura a tre livelli

Figura 15

Architettura client-server a due livelli

modulo che realizza il client è specifico dell'applicazione: quando la logica di elaborazione cambia, bisogna distribuire a tutti i calcolatori il nuovo client. Al contrario, in un'applicazione a tre livelli la logica di elaborazione risiede sul server: quando tale logica viene modificata, si aggiorna il codice del server, mentre il livello di presentazione (il browser) rimane identico. Ciò è molto più semplice da gestire, rispetto all'aggiornamento di molti calcolatori.

Tabella 1

Tabella CityZone

Città	***Fuso orario***
San Francisco	*America/Los_Angeles*

Kaohsiung
Asia/Taipei
...
...

Nel nostro esempio, useremo una base di dati con un'unica tabella, CityZone, contenente i nomi delle città e dei fusi orari (osservate la Tabella 1). Quando l'utente inserisce il nome di una città, il JavaBean ZoneDBBean consulta, per prima cosa, la base di dati, cercando un nome di fuso orario che corrisponda alla città desiderata. Se non trova una tale corrispondenza, effettua una ricerca negli identificativi di fuso orario disponibili, come negli esempi precedenti.

File cityzone.sql

CREATE TABLE CityZone (City CHAR(30), Zone CHAR(45))
INSERT INTO CityZone
VALUES ('San Francisco', 'America/Los_Angeles')

INSERT INTO CityZone
VALUES ('Kaohsiung', 'Asia/Taipei')
...

Per interrogare la base di dati, il JavaBean ha bisogno di un oggetto di tipo Connection, che si crea invocando il metodo statico getConnection della classe DriverManager:
Connection conn = DriverManager.getConnection(
url, username, password);

Dove va inserito questo codice? Non vogliamo assolutamente dover memorizzare il nome utente e la parola d'accesso per la base di dati all'interno della classe ZoneDBBean (e in tutte le altri classi che usano la base di dati). Al contrario, usiamo un'altra classe, DataSourceBean, il cui solo scopo è quello di gestire le connessioni alla base di dati. L'implementazione di tale classe è molto semplice: ci sono quattro metodi per impostare il driver da utilizzare, lo URL della base di dati, il nome utente e la parola

d’accesso, nonché un metodo getConnection che restituisce un oggetto che rappresenta una connessione.

Un JavaBean con visibilità di ap-

Ci serve un unico esemplare della classe DataSourceBean per servire plicazione viene inizializzato una

tutte le pagine JSP che compongono l’applicazione Web; per raggiun-sola volta ed è attivo per tutta la

gere questo scopo, impostiamo la visibilità del JavaBean al valore “ap-vita dell’applicazione Web.

plication”:

<jsp:useBean id=”db” class=”DataSourceBean”

scope=”application”/>

Dove dovremmo inserire i parametri di accesso alla base di dati? Codificarli all’interno della classe DataSourceBean sarebbe un errore. Nel capitolo dedicato alla programmazione delle basi di dati, abbiamo usato un file contenente le proprietà della base di dati, ma usare un file in un’applicazione JSP è scomodo, perché un JavaBean non sa come localizzare i nomi di file relativi, mentre non vogliamo assolutamente usare un nome di file assoluto. Fortunatamente, i progettisti della tecnologia JSP hanno previsto problemi come questo e hanno messo a disposizione un modo migliore per fornire informazioni a un’applicazione Web.

Quando viene fatto partire, il motore JSP legge un file di impostazioni avente il nome web.xml e posto all’interno della sottocartella WEB-INF, a sua volta contenuta nella cartella dell’applicazione Web. Questo file è il posto giusto per memorizzare parametri di configurazione, come le proprietà di una base di dati, e ha un formato XML, con la struttura seguente:

<?xml version=”1.0”/>

<!DOCTYPE web-app

PUBLIC

“-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN”

“http://java.sun.com/j2ee/dtds/web-app_2_3.dtd”>
<web-app>
parametri di configurazione
altre opzioni
</web-app>

Notate che il DOCTYPE del vostro motore JSP può essere diverso.

I parametri di configurazione che riguardano tutti i file JSP dell’applicazione Web sono contenuti in elementi di tipo <context-param>; ciascun parametro ha un nome e un valore. Ad esempio, ecco come si possono scrivere le proprietà di una base di dati:

```
<context-param>
<param-name>jdbc.drivers</param-name>
<param-value>COM.cloudscape.core.JDBCDriver</param-value>
</context-param>
<context-param>
<param-name>jdbc.url</param-name>
<param-value>jdbc:cloudscape:InvoiceDB;create=true</param-value>
</context-param>
<context-param>
<param-name>jdbc.username</param-name>
<param-value>username</param-value>
</context-param>
<context-param>
<param-name>jdbc.password</param-name>
<param-value>password</param-value>
</context-param>
```

Per accedere ai valori dei parametri da una pagina JSP, usate il metodo getInitParameter dell’oggetto predefinito application, che viene definito in ogni pagina JSP, in modo

simile all'oggetto request che avete già utilizzato, ed è un esemplare della classe ServletContext. Ad esempio, potete ottenere il nome utente per la base di dati in questo modo:

<%= application.getInitParameter("jdbc.username") %>

Potete inserire i parametri di ini-Per inizializzare il JavaBean che gestisce le connessioni alla base di dati, zializzazione nel file web.xml e

avete bisogno di recuperare questi valori, ma c'è una piccola complica-accedervi tramite l'oggetto pre-

zione. Vogliamo che l'inizializzazione avvenga una volta sola, quando il definito application.

JavaBean viene costruito per la prima volta, e non ogni volta che si accede alla pagina. Questa è una situazione comune e la direttiva jsp:useBean ha un meccanismo per gestirla: qualsiasi direttiva che venga inserita come figlia delle direttive jsp:useBean viene eseguita una sola volta, quando il JavaBean viene costruito. Ecco le direttive per costruire il JavaBean che gestisce le connessioni alla base di dati:

```
<jsp:useBean id="db" class="DataSourceBean" scope="application">
<jsp:setProperty name="db" property="driver" value=
"<%= application.getInitParameter(\"jdbc.drivers\") %>"/>
<jsp:setProperty name="db" property="url" value=
"<%= application.getInitParameter(\"jdbc.url\") %>"/>
<jsp:setProperty name="db" property="username" value=
"<%= application.getInitParameter(\"jdbc.username\") %>"/>
<jsp:setProperty name="db" property="password" value=
"<%= application.getInitParameter(\"jdbc.password\") %>"/>
</jsp:useBean>
```

JavaServer Pages

Come già detto nel capitolo relativo alla programmazione delle basi di dati, i programmi professionali usano un metodo più sofisticato per gestire le connessioni a una base di dati, le connessioni condivise (pooled connections). Quando una connessione con-divisa viene chiusa, non termina fisicamente, ma viene rimessa in una coda e nuovamente restituita come risultato di una successiva invocazione del metodo getConnection. Tuttavia, mentre si sta scrivendo questo libro, non esiste un meccanismo standard per la condivisione delle connessioni né in Java, né nella tecnologia JSP, per cui dovrete cercare una sorgente di dati dotata di connessioni condivise. Sia i produttori di basi di dati che di contenitori JSP spesso le forniscono e potete trovare molte implementazioni, di varia qualità, su Internet.

La classe ZoneDBBean è una variante della classe TimeZoneBean che, prima di cercare un fuso orario tra gli identificativi standard, effettua un'interrogazione della base di dati:

SELECT Zone FROM CityZone WHERE City = la città richiesta Se esiste nella base di dati un valore che corrisponde, viene restituito tale fuso orario.

Ecco il codice completo per questa applicazione Web:

File zonedb.html

```
<html>
<head>
<title>Time Zone Database Form</title>
</head>
<body>
<form action="zonedb.jsp" method="POST">
<p>City:
<input type="text" name="city"/>
<input type="submit" value="Get Time"/>
</p>
</form>
</body>
```

```
</html>
```

File zonedb.jsp

```
<jsp:useBean id="db" class="DataSourceBean"
scope="application">
<jsp:setProperty name="db" property="driver" value=
"<%= application.getInitParameter(\"jdbc.drivers\")
%>"/>
<jsp:setProperty name="db" property="url" value=
"<%= application.getInitParameter(\"jdbc.url\") %>"/>
<jsp:setProperty name="db" property="username"
value=
"<%= application.getInitParameter(\"jdbc.username\")
%>"/>
<jsp:setProperty name="db" property="password"
value=
"<%= application.getInitParameter(\"jdbc.password\")
%>"/>
</jsp:useBean>
<jsp:useBean id="zone" class="ZoneDBBean"/>
<jsp:setProperty name="zone" property="city"
param="city"/>
<jsp:setProperty name="zone" property="connection"
```



```
value="<%= db.getConnection() %>"/>
<jsp:setProperty name="zone" property="date"
value="<%= new java.util.Date() %>"/>
<html>
<head>
<title>Zone Database JSP</title>
</head>
<body>
<h1>Zone Database JSP</h1>
<p>The current time in
<%= request.getParameter("city") %>
is:
```

```
<jsp:getProperty name="zone" property="time"/>
</p>
</body>
</html>
```

File DataSourceBean.java

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;
/**
Un JavaBean che rappresenta una semplice sorgente di dati per stabilire connessioni a una base di dati.
*/
public class DataSourceBean
{
/**
Proprietà di sola scrittura, "driver".
@param driver il nome del driver della base di dati
*/
public void setDriver(String driver)
throws ClassNotFoundException
{
Class.forName(driver);
}
/**
Proprietà di sola scrittura, "url".
@param aUrl lo URL JDBC
*/
public void setUrl(String aUrl)
{
url = aUrl;
}
/**
Proprietà di sola scrittura, "username".
@param aUsername il nome utente per la base di dati
*/
```

JavaServer Pages

```
public void setUsername(String aUsername)
{
username = aUsername;
}
/**
Proprietà di sola scrittura, “password”.
@param aPassword la parola d’accesso per la base di dati
*/
public void setPassword(String aPassword)
{
password = aPassword;
}
/**
Fornisce una connessione alla base di dati.
@return una connessione alla base di dati
*/
public static Connection getConnection()
throws SQLException
{
return DriverManager.getConnection(
url, username, password);
}
private static String url;
private static String username;
private static String password;
}
```

File ZoneDBBean.java

```
import java.text.DateFormat;
import java.util.Date;
import java.util.TimeZone;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
/**
Questo JavaBean compone in una stringa l'orario corrispondente a un certo istante in una certa città, consultanto una base di dati oltre all'elenco degli identificativi standard per i fusi orari.
*/
public class ZoneDBBean
{
/**
Inizializza il compositore di stringhe.
*/
public ZoneDBBean()
{
timeFormatter = DateFormat.getTimeInstance();
}
```



```
/**
Proprietà "city".
@return la città per la quale si richiede l'orario
*/
public String getCity()
{
return city;
}
/**
Proprietà "city".
@param aCity la città per la quale si richiede l'orario
*/
public void setCity(String aCity)
{
city = aCity;
}
/**
Proprietà di sola scrittura, "date".
@param aDate l'orario da utilizzare
```

```
*/
public void setDate(Date aDate)
{
theDate = aDate;
}
/**
Proprietà di sola lettura, “time”.
@return l’orario composto come stringa
*/
public String getTime()
{
TimeZone zone = getTimeZoneDB();
if (zone == null) // fuso orario non presente nella base di dati zone = getTimeZone(city);
if (zone == null) return “not available”;
timeFormatter.setTimeZone(zone);
return timeFormatter.format(theDate);
}
/**
Cerca il fuso orario di una città nella base di dati e nell’elenco degli identificativi dei fusi orari.
@return il fuso orario, oppure null se non lo si è trovato
*/
private TimeZone getTimeZoneDB() throws SQLException
{
Connection conn = null;
try
{
conn = DataSourceBean.getConnection();
Statement stat = conn.createStatement();
String query = “SELECT Zone FROM CityZone”
```


```
+ “ WHERE City = ‘” + city + “’”;
ResultSet result = stat.executeQuery(query);
if (result.next())
```

```
return TimeZone.getTimeZone(
result.getString(1));
else
return null;
}
finally
{
if (conn != null) conn.close();
}
}
/**
Cerca il fuso orario di una città nell'elenco degli identificativi dei fusi orari.
@param aCity la città per cui si cerca il fuso orario
@return il fuso orario, oppure null se non lo si è trovato
*/
private static TimeZone getTimeZone(String city)
{
String[] ids = TimeZone.getAvailableIDs();
for (int i = 0; i < ids.length; i++)
if (timeZoneIDmatch(ids[i], city))
return TimeZone.getTimeZone(ids[i]);
return null;
}
/**
Verifica se un fuso orario corrisponde a una città.
@param id la stringa che identifica il fuso orario (ad esempio, "America/Los_Angeles")
@param aCity la città per cui si cerca il fuso orario (ad esempio, "Los Angeles")
@return true se c'è corrispondenza
*/
private static boolean timeZoneIDmatch(String id, String city)
{
```

String idCity = id.substring(id.indexOf('/') + 1); return idCity.replace('_', ' ').equals(city);
}
private DateFormat timeFormatter;
private Date theDate;
private String city;
}
JavaServer Pages
45
8
Servlet
Nei paragrafi precedenti avete visto come usare la tecnologia JSP per costruire applicazioni Web: una pagina JSP specifica il codice HTML statico e le informazioni dinamiche che l'utente dell'applicazione Web vede nel suo browser, e fornisce anche una connessione ai JavaBean che svolgono le elaborazioni.
I servlet Java sono un meccanismo alternativo per costruire applica-Un servlet è un programma Java
che svolge elaborazioni e gene-
zioni Web. Un servlet è un programma Java che svolge elaborazioni, ra dati che vengono restituiti al
alcune delle quali generano codice HTML. Come vedrete nel prossimo browser.
paragrafo, i servlet e la tecnologia JSP sono strettamente correlati: ciascuna pagina JSP viene automaticamente tradotta in un servlet.
Quando scrivete un servlet, fornite una classe che estende la classe HttpServlet e realizza uno dei metodi seguenti, o entrambi:
void doGet(HttpServletRequest request,
HttpServletResponse response)
void doPost(HttpServletRequest request,
HttpServletResponse response)
in relazione al fatto che vogliate che il vostro servlet gestisca le richieste di tipo GET, di tipo POST, o entrambe.

Per il nostro primo servlet, elaboreremo soltanto richieste di tipo GET. In realtà, non invieremo nessuna informazione insieme alla richiesta: il servlet creerà semplicemente una pagina HTML con l'ora attuale e la invierà in risposta al browser.

Il parametro request contiene dettagli relativi alla richiesta: useremo tale classe nell'esempio successivo. In questo semplice servlet, l'informazione restituita non dipende da dati forniti dall'utente, per cui non ci occupiamo della richiesta.

Il parametro response vi consente di specificare cosa viene inviato in risposta al browser. Se volete inviare un documento HTML (il caso più comune), dovete fare come segue:

1. Impostate il tipo del contenuto a HTML:

response.setContentType("text/html");

2. Ottenete un oggetto di tipo PrintWriter che raccolga il documento che deve essere inviato al browser:

PrintWriter out = response.getWriter();

3. Scrivete il documento HTML nell'oggetto di tipo PrintWriter.

4. Chiudete l'oggetto di tipo PrintWriter.

Ecco il servlet completo. Notate il codice che genera il documento HTML scrivendo i marcatori (come <html>, <head> e così via) insieme alle informazioni per l'utente. Nel nostro caso, l'informazione è semplicemente costituita dall'ora e data attuali.

File DateServlet.java

```
import java.io.PrintWriter;
import java.io.IOException;
import java.util.Date;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
/**
Questo servlet stampa l'ora e la data attuali.
*/
public class DateServlet extends HttpServlet
{
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
{
// ottieni l'informazione
String now = new Date().toString();
// imposta il tipo del contenuto a HTML
response.setContentType("text/html");
// stampa l'informazione impaginata
PrintWriter out = response.getWriter();
String title = "Date Servlet";
out.println("<html><head><title>"); out.println(title);
out.println("</title></head><body><h1>");
out.println(title);
out.println(</h1><p>The current time is: ");
out.println(now);
out.println("</p></body></html>"); out.close();
}
}
```

La Figura 16 mostra ciò che viene prodotto dal servlet.

Per caricare un servlet in un ser-

Come potete vedere, la struttura di base di un servlet è semplice: ver Web dovete fornire il suo de-dovete recuperare l'informazione richiesta dall'utente e restituire il co-scrittore di caricamento.

dice HTML per visualizzare la risposta in un modo piacevole e efficace.

Caricare un servlet nel server Web è un po' più complicato di quanto fatto per una pagina JSP. Dovete scrivere un descrittore di caricamento, in cui fornite il nome

del vostro servlet, che può essere diverso dal nome della classe, e specificate due altre informazioni:

- *Come trovare la classe del servlet*
- *Quali sono gli URL che sono gestiti dal servlet, cioè che provocano la sua invocazione*

Figura 16

Il prodotto del servlet DateServlet

Ecco un esempio di descrittore di caricamento per il nostro servlet che fornisce la data e l'ora attuali, che abbiamo chiamato Date. Il descrittore di caricamento dice al motore dei servlet che tale servlet è implementato dalla classe DateServlet e che uno URL con suffisso /date dovrebbe attivare il servlet.

Il descrittore di caricamento deve trovarsi nel file Web.xml. Notate che la descrizione del DOCTYPE potrebbe essere diversa, nel vostro sistema.

File web.xml

```
<?xml version="1.0"/>
<!DOCTYPE web-app
PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
```

“http://java.sun.com/j2ee/dtds/web-app_2_3.dtd”>
<web-app>
<servlet>
<servlet-name>Date</servlet-name>
<servlet-class>DateServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Date</servlet-name>
<url-pattern>/date</url-pattern>
</servlet-mapping>
</web-app>

Il file web.xml deve essere collocato nella sottocartella WEB-INF della cartella che contiene la vostra applicazione Web, come, ad esempio: c:\jakarta-tomcat\webapps\bigjava\WEB-INF

Figura 17
Posizionamento dei file di un’applicazione Web

Infine, dovete copiare i file con le classi nella sottocartella classes della cartella WEB-INF, esattamente come avete fatto per i file con le classi dei JavaBean (osservate la Figura 17).

Nel file web.xml, siete liberi di scegliere qualsiasi nome per il servlet e per il suffisso dello URL. Tale flessibilità non è, in realtà, necessaria, per cui può darsi che vogliate dare a

entrambi il nome della classe, oppure potete usare nomi diversi, come nell'esempio precedente.

Lo URL completo per eseguire il servlet è il nome dell'applicazione Web seguito dal suffisso, ad esempio:

http://localhost:8080/bigjava/date

Ora diamo un'occhiata al secondo servlet, che consente all'utente di specificare il nome di una città. In questo servlet usiamo il parametro request del metodo doPost per ottenere il valore fornito dall'utente: questo parametro è identico alla variabile request di una pagina JSP. Per recuperare un parametro dalla richiesta, usate il metodo getParameter:

```
String cityName = request.getParameter("city");
```

File zoneservlet.html

```
<html>
<head>
<title>Time Zone Servlet Form</title>
</head>
<body>
<form action="timezone" method="POST">
<p>City:
<input type="text" name="city"/>
```



```
<input type="submit" value="Get Time"/>
</p>
</form>
</body>
</html>
```

File TimeZoneServlet.java

```
import java.io.PrintWriter;
import java.io.IOException;
import java.text.DateFormat;
import java.util.Date;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
Questo servlet stampa l'ora e la data attuali
con il fuso orario della città specificata come valore del
parametro "city"
*/
public class TimeZoneServlet extends HttpServlet
{
public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
{
// ottieni l'informazione
String cityName = request.getParameter("city");
TimeZone zone = getTimeZone(cityName);
// imposta il tipo del contenuto a HTML
response.setContentType("text/html");
// stampa l'informazione impaginata
PrintWriter out = response.getWriter();
String title = "Time Zone Servlet";
out.println("<html><head><title>"); out.println(title);
out.println("</title></head><body><h1>");
out.println(title);
out.println(</h1><p>");
if (zone == null)
out.println("Sorry–unknown city");
else
{
out.print("The current time in <b>");
out.print(cityName);
out.print("</b> is: ");
DateFormat formatter =
DateFormat.getTimeInstance();
formatter.setTimeZone(zone);
```

JavaServer Pages

```
Date now = new Date();
out.print(formatter.format(now));
}
out.println("</p></body></html>"); out.close();
}
/**
Cerca il fuso orario di una città nell'elenco degli identificativi dei fusi orari.
@param aCity la città per cui si cerca il fuso orario
@return il fuso orario, oppure null se non lo si è trovato
*/
private static TimeZone getTimeZone(String city)
{
String[] ids = TimeZone.getAvailableIDs();
for (int i = 0; i < ids.length; i++)
if (timeZoneIDmatch(ids[i], city))
return TimeZone.getTimeZone(ids[i]);
return null;
}
/**
Verifica se un fuso orario corrisponde a una città.
@param id la stringa che identifica il fuso orario (ad esempio, "America/Los_Angeles")
@param aCity la città per cui si cerca il fuso orario (ad esempio, "Los Angeles")
@return true se c'è corrispondenza
*/
private static boolean timeZoneIDmatch(String id, String city)
{
String idCity = id.substring(id.indexOf('/') + 1); return idCity.replace('_', ' ').equals(city);
}
}
```

Con la sola differenza della gestione del parametro, questo servlet è simile al primo esempio di servlet. Tuttavia, se osservate il codice del programma, noterete che non è facile capire come appaia il risultato prodotto dal servlet: dovete verificare attentamente cosa viene scritto nel flusso out. Al contario, una pagina JSP permette di vedere molto facilmente come appaia il risultato prodotto. La tecnologia JavaServer Pages incoraggia anche la separazione di presentazione ed elaborazione, molto più di quanto faccia la tecnologia dei servlet.

Per questa ragione, le linee guida Java 2 per lo sviluppo d'impresa raccomandano l'uso di JavaServer Pages come tecnologia principe per la produzione di interfacce Web. Allora, perché mai occuparsi di servlet? I servlet hanno un loro ruolo, ad esempio, nel caso in cui il risultato di un'interrogazione Web non sia costituito da codice HTML ma da dati in formato binario, come un immagine o un file di programma da scaricare. Inoltre, come vedrete nel prossimo paragrafo, la pagine JSP vengono com-JavaServer Pages

pilate e trasformate in servlet: conoscere le caratteristiche principali dei servlet può aiutarvi a capire come scrivere pagine JSP.

Errori comuni 3

Variabili istanza nei servlet

Se guardate con attenzione al codice della classe TimeZoneServlet, noterete che il metodo doPost fornisce i dati da impaginare a vari metodi statici. Sarebbe stato possibile usare una variabile istanza per il campo city, invece di passarlo come parametro ai metodi? Questa idea, apparentemente sensata, non funziona con i servlet. Il contenitore di servlet costruisce un unico oggetto servlet, che risponde a tutte le richieste. Se giungono simultaneamente diverse richieste, vengono eseguite in parallelo, in thread separati, ma tutte le richieste condividono lo stesso oggetto e, quindi, le stesse variabili

istanza. Di conseguenza, potete usare variabili istanza soltanto per quelle informazioni che volete condividere tra diverse richieste al servlet. Poiché i nomi di città sono diversi per ciascuna richiesta, non potete memorizzarle in variabili istanza.

Le variabili locali sono, invece, sicure: ciascun thread usa una diversa copia delle variabili locali.

9

Compilazione di pagine JSP

Quando scrivete una pagina JSP, il motore JSP la traduce in un servlet nel momento in cui giunge la prima richiesta o quando la pagina è stata modificata ri-Il contenitore JSP traduce le pa-

gine JSP in servlet.

spetto all'ultima richiesta ricevuta. Può darsi che abbiate notato che, a volte, occorre un po' di tempo perché una pagina JSP invii la risposta al browser: il ritardo è provocato da questo processo di traduzione. In un ambiente operativo, questo non è un problema, perché una pagina JSP, dopo essere stata collauda-ta, rimane tale e quale per lungo tempo.

Considerate il nostro primo esempio di JSP, la pagina che fornisce la data e l'ora attuale:

```
<html>
<head>
<title>Date JSP</title>
</head>
<body>
<h1>Date JSP</h1>
<p>The current time is:
<%= new java.util.Date() %>
</p>
</body>
</html>
```

Questa pagina viene tradotta in un servlet avente il seguente metodo doPost: public void doPost(HttpServletRequest request,

```
HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println(
"<html><head><title>Date JSP</title></head>"
+ "<body><h1>Date JSP</h1>"
+ "<p>The current time is:");
out.println(new java.util.Date());
out.println("</p></body></html>"); out.close();
}
```

Anche le direttive per i JavaBean vengono tradotte in codice Java. Ad esempio:

```
<jsp:setProperty name="zone" property="city"
param="city"/>
```

diventa

```
zone.setCity(request.getParameter("city"));
```

Ecco, invece, come viene tradotto il codice per la diramazione e il rimando del Paragrafo 6.

```
<%
if (zone.isAvailable())
{
%>
<jsp:forward page="zoneresult.jsp"/>
<%
}
else
{
%>
<jsp:forward page="zoneerror.jsp"/>
<%
}
```

```
%>
diventa
if (zone.isAvailable())
{
out.clear();
pageContext.forward("zoneresult.jsp");
return;
}
else
```



```
{
out.clear();
pageContext.forward("zoneerror.jsp");
return;
}
```

A questo punto potete capire perché siano necessarie le parentesi graffe attorno alle direttive jsp:forward: ciascuna direttiva viene tradotta con più di un enunciato Java.

Questa panoramica del processo di compilazione di JSP termina il capitolo sulla tecnologia JavaServer Pages. Ora, avete visto come sviluppare applicazioni in Java eseguite sul lato server e come ottenere la separazione della presentazione (il codice HTML

nelle pagine JSP) dalla logica di elaborazione (il codice Java nei JavaBean).

Consigli per la produttività 1

Esaminare il codice del servlet

A dire il vero, è possibile esaminare i file sorgente dei servlet in cui vengono tradotti i file JSP: essi si trovano nella sottocartella work della cartella del motore JSP, ad esempio c:\jakarta-tomcat\work. Ciò può essere abbastanza utile per localizzare gli errori nelle vostre pagine JSP.

Ecco un tipico servlet: il risultato della compilazione della pagina multizone.jsp.

Parte del codice è stato rimosso, per semplicità; inoltre, per impaginare il file, sono state inserite delle interruzioni di riga e alcune stringhe sono state scomposte.

```
import enunciati eliminati
public class _0002fmultizone_0002ejspmultizone_jsp_1
extends HttpJspBase
{
alcuni metodi standard eliminati
public void _jspService(HttpServletRequest request,
HttpServletResponse response)
throws IOException, ServletException
{
codice standard eliminato
MultiZoneBean zones =
(MultiZoneBean)pageContext.getAttribute(
“zones”, PageContext.SESSION_SCOPE);
pageContext.setAttribute(
“zones”, zones, PageContext.SESSION_SCOPE);
JspRuntimeLibrary.handleSetProperty(
pageContext.findAttribute(“zones”),
“date”, new java.util.Date());
JspRuntimeLibrary.introspecthelper(
pageContext.findAttribute(“zones”), “city”,
```



```
request.getParameter(“city”), request,
“city”, false);
JspRuntimeLibrary.introspecthelper(
pageContext.findAttribute(“zones”), “city”,
“<jsp:getProperty name=\”zones\””
+ “ property=\”times\”/>”,
null, null, false);
out.write(
“\n\n<html>\n<head>\n”
+ “<title>Multiple Time Zone JSP</title>\n”
+ “</head>\n<body>\n”
```

+ “<h1>Current time lookup</h1>\n<ul>\n”); out.print(JspRuntimeLibrary.toString(
(((MultiZoneBean)pageContext.findAttribute(
“zones”)).getTimes())));
*out.write(“\n
\n”*
+ “<form action=\”multizone.jsp\””
+ “ method=\”POST\”>\n Next City:\n”
+ “<input type=\”text\” name=\”city\”/>\n”
+ “<input type=\”submit\” value=\”Add City\”/>”
+ “\n \n</form>\n</body>\n</html>\n”); codice standard eliminato
}
}

Ovviamente, gran parte del codice sarà misterioso, ma a volte è possibile vedere perché un certo costrutto JSP non sembra compilarsi nel modo corretto.

Note di cronaca 1

Catalogare la propria collezione di cravatte

Persone e aziende utilizzano i computer per organizzare praticamente tutti gli aspetti delle nostre vite. Nel complesso, i computer sono eccezionalmente utili per acquisire e analizzare dati. In effetti, la potenza messa a disposizione dai computer e dal loro software ne fa soluzioni molto attraenti per qualsiasi problema organizzativo. È facile perdere di vista il fatto che il ricorso a un computer non è sempre la miglior soluzione di un problema.

Nel 1983, l’autore John Bear raccontò di una persona che aveva ideato un nuovo utilizzo per i computer domestici che si erano recentemente resi disponibili. Quel signore catalogava la propria collezione di cravatte, inserendo le descrizioni delle cravatte in una base di dati e generando rapporti che le elencano per colore, prezzo o stile. C’è da sperare che quella persona avesse trovato qualche altro utilizzo per giusti-ficare l’acquisto di un’apparecchiatura che costa comunque qualche milione, però quell’applicazione era così cara al suo cuore che voleva condividerla con altri.

Non c'è da stupirsi che siano pochi gli utenti che condividono quell'interesse e non troverete gli scaffali del vostro negozio di software pieni di programmi per catalogare le cravatte.

JavaServer Pages

55

Il fenomeno dell'uso della tecnologia tanto per il gusto di farlo è molto diffuso.

Verso la metà degli anni Novanta, parecchie grandi aziende dimostrarono un grande entusiasmo per l'utilizzo di reti di computer per consegnare a richiesta film a utenti domestici. Con la tecnologia del giorno d'oggi, questo è un metodo molto costoso per far arrivare un film a casa di qualcuno. Occorrono connessioni di rete ad alta velocità e nuove apparecchiature di ricezione. Sembra un impegno sproporzionato per eliminare la passeggiata fino al più vicino negozio di noleggio di film. E, in effetti, i primi esperimenti sul campo ebbero un effetto scoraggiante. In questi esperimenti, le linee di rete e i computer venivano simulati da persone che andavano a inserire nastri in videolettori remoti. Pochi clienti erano disposti a pagare per questo servizio un prezzo adeguato che giustificasse gli enormi investimenti che sarebbero stati necessari. In futuro potrebbe anche rivelarsi conveniente, dal punto di vista economico, inviare film attraverso reti di computer, ma oggi come oggi gli economici lettori di videocas-sette e i film noleggiati rendono un servizio migliore a un costo inferiore.

Nella "esplosione di Internet" del 2000, sono state fondate centinaia di aziende, con la speranza che Internet rendesse tecnologicamente possibile ordinare con un computer domestico articoli come alimentari e cibo per animali, sostituendo i negozi tradizionali con negozi sul Web. Di nuovo, la realizzabilità tecnologica non assicurò il successo economico. Trasportare i generi alimentari e il cibo per animali fino a casa dei clienti non era abbastanza

economico e pochi clienti erano disposti a pagare un prezzo addizionale per tale comodità.

Nel periodo in cui questo libro viene scritto, molte scuole elementari negli Stati Uniti stanno spendendo enormi quantità di denaro per portare i computer e Internet nelle aule scolastiche. In realtà, è facile capire perché insegnanti, presidi, genitori e politici siano favorevoli ai computer nelle aule scolastiche. Non è forse l'alfabetizza-zione informatica qualcosa di assolutamente essenziale per i giovanissimi nel nuovo millennio? Non è forse particolarmente importante dare ai ragazzi provenienti da famiglie a basso reddito, i cui genitori forse non si possono permettere l'acquisto di un computer domestico, l'opportunità di impadronirsi delle competenze informatiche?

Le scuole, però, hanno scoperto che i computer sono enormemente costosi. Il costo iniziale per l'acquisto dell'apparecchiatura, ancorché rilevante rispetto al costo dei libri e di altri materiali didattici, non è fuori della portata dei bilanci della maggior parte delle scuole. Quello che è devastante, però, è il costo totale del possesso (total cost of ownership, TCO): vale a dire, il costo iniziale più il costo che si deve sostenere per tenere i computer in buone condizioni e per sostituirli quando diventano obsoleti. A mano a mano che le scuole hanno acquistato più apparecchiature di quante potessero essere tenute in esercizio da qualche occasionale volontario, hanno dovuto fare scelte molto dolorose; licenziare bibliotecari e assistenti di storia dell'arte per assumere un maggior numero di tecnici informatici oppure lasciare che le costose apparecchiature diventassero inutilizzabili? E quali erano, alla fin fine, i veri vantaggi educativi? La cosa interessante è che molte scuole si sono lasciate talmente trascinare dall'entusiasmo che non si sono neppure poste questa domanda, se non dopo che i computer erano arrivati.

Come programmatori di computer potremmo essere inclini a programmare qualsiasi cosa. Come professionisti

del computer, però, abbiamo, nei confronti dei nostri datori di lavoro e dei nostri clienti, il dovere di capire quali problemi vogliono risolvere e installare computer e software soltanto quando aggiungono un valore superiore al loro costo.

Riepilogo del capitolo

1. Una pagina JavaServer Pages (JSP) contiene marcatori HTML e istruzioni Java.

2. Il contenitore JSP rimuove tutti i marcatori JSP e inserisce al loro posto le stringhe prodotte dalla valutazione delle espressioni Java.

3. Un JavaBean è una classe con un costruttore predefinito che mette a disposizione le proprie caratteristiche mediante metodi get e set.

4. Una pagina JSP vi consente di accedere alle proprietà di un JavaBean senza scrivere codice Java.

5. Usate i JavaBean per separare la presentazione HTML dai calcoli in Java.

6. Quando un browser invia un form a un server Web, invia i nomi e i valori di tutti gli elementi del form.

7. In una pagina JSP potete accedere ai dati del form attraverso l'oggetto predefinito request.

8. Per fornire a un JavaBean un parametro proveniente dalla richiesta, esiste una comoda abbreviazione.

9. Un form HTML contiene elementi dell'interfaccia utente, come campi di testo e pulsanti.

10. Un form HTML deve specificare lo URL del programma che, sul server, elabora i dati del form stesso.

11. Una sessione è una sequenza di pagine richieste da uno stesso browser a uno stesso server Web.

12. Per gestire una sessione in una sequenza di pagine JSP, usate un JavaBean con visibilità di sessione.

13. Una pagina JSP può inviare a un'altra pagina la richiesta ricevuta, una tecnica utile per realizzare

diramazioni.

14. Un JavaBean con visibilità di tipo richiesta (request) rimane attivo quando la richiesta viene inviata di rimando a un'altra pagina.

15. Un'applicazione a tre livelli ha livelli distinti per la presentazione, l'elaborazione e la memorizzazione dei dati.

16. Un JavaBean con visibilità di applicazione viene inizializzato una sola volta ed è attivo per tutta la vita dell'applicazione Web.

17. Potete inserire i parametri di inizializzazione nel file Web.xml e accedervi tramite l'oggetto predefinito application.

18. Un servlet è un programma Java che svolge elaborazioni e genera dati che vengono restituiti al browser.

19. Per caricare un servlet in un server Web dovete fornire il suo descrittore di caricamento.

20. Il contenitore JSP traduce le pagine JSP in servlet.

Ulteriori letture

[1] http://jakarta.apache.org/tomcat/index.html La pagina Web del server Apache Tomcat.

[2] Marry Hall, Core Servlets and JavaServer Pages, Sun Microsystems Press/
Prentice Hall, 2000.

[3] Damon Hougland e Aaron Tavistock, Core JSP, Prentice Hall, 2001.

Classi, oggetti e metodi
presentati nel capitolo

java.util.DateFormat
format
getTimeInstance
setTimeZone
java.util.TimeZone
getTimeZone
getAvailableIDs

javax.servlet.ServletContext
getInitParameter
javax.servlet.http.HttpServlet
doGet
doPost
javax.servlet.http.HttpServletRequest
getParameter
javax.servlet.http.HttpServletResponse
setContentType
getWriter

Esercizi di ripasso

Esercizio R.1. *Qual è la differenza tra una pagina JSP e un contenitore JSP?*

Esercizio R.2. *Cos'è un JavaBean?*

Esercizio R.3. *Cos'è una proprietà di un JavaBean?*

Esercizio R.4. *Un oggetto di tipo JButton è un JavaBean? Perché, oppure perché no?*

Esercizio R.5. *Dal punto di vista dell'ingegneria del software, qual è lo scopo dell'utilizzo dei JavaBean insieme alle pagine JSP?*

Esercizio R.6. *La tecnologia JSP offre ai JavaBean due servizi che li rendono più flessibili delle variabili Java: la visibilità e l'inizializzazione con i parametri della richiesta.*

Fornite una spiegazione.

Esercizio R.7. *Quali sono le scelte possibili per la visibilità di un JavaBean? Qual è la visibilità predefinita? Quando si dovrebbe scegliere ciascuno dei vari ambiti visibilità?*

Esercizio R.8. *Come potete realizzare il controllo d'errore in un form usando le pagine JSP? Fornite spiegazioni, usando come esempio l'autenticazione di un utente (login).*

Esercizio R.9. *Quali informazioni invia un browser a una applicazione Web quando invia un form? Come può*

l'applicazione analizzare tali informazioni?

Esercizio R.10. *Quali elementi potete inserire in un form HTML per ricevere dati in ingresso? Quali sono i loro equivalenti Swing?*

Esercizio R.11. *Cos'è una sessione? Perché le applicazioni Web hanno bisogno di un meccanismo per gestire le sessioni? Perché ciò non viene fornito dal protocollo HTTP?*

Esercizio R.12. *Qual è la differenza tra un'applicazione client-server e un'applicazione a tre livelli?*

Esercizio R.13. *In un'applicazione Web, come potete memorizzare e recuperare i parametri di inizializzazione senza scriverli all'interno di una pagina JSP o di un JavaBean?*

Esercizio R.14. *Che relazione esiste tra pagine JSP e servlet?*

Esercizio R.15. *Nel vostro motore JSP, trovate il codice del servlet generato dalla compilazione della pagina multizone.jsp. Dove l'avete trovato? Qual è il codice Java per costruire un JavaBean con visibilità di sessione?*

Esercizi di programmazione

Esercizio P.1. *Scrivete una pagina JSP che riporti i valori delle seguenti proprietà di sistema relative al server Web:*

- *La versione di Java (java.version)*
- *Il nome del sistema operativo (os.name)*
- *La versione del sistema operativo (os.version) Usate il metodo getProperties della classe System.*

Esercizio P.2. *Scrivete una pagina JSP che simuli il lancio di due dadi, producendo un'informazione del tipo "Ottenuto un 4 e un 6". Quando l'utente ricarica la pagina, viene visualizzata una nuova coppia di valori. Usate il metodo Math.random per evitare di dover creare un oggetto di tipo Random.*

Esercizio P.3. *Migliorate l'esercizio precedente generando una pagina Web che mostri immagini dei dadi*

lanciati. Cercate immagini GIF delle facce di un dado con nu-

meri da 1 a 6 e generate una pagina HTML che faccia riferimento alle immagini ap-propriate.

Esercizio P.4. *Aggiungete la proprietà format alla classe LotteryBean vista in Consigli pratici 1. Se format vale "ul", viene prodotto un elenco puntato, mentre se vale*

"table" viene generata una tabella HTML.

Esercizio P.5. *Suddividete la classe LotteryBean vista in Consigli pratici 1 in due JavaBean. Il primo restituisce un vettore di oggetti di tipo Integer che incapsulano i numeri estratti, mentre il secondo compone un vettore qualsiasi in un elenco puntato HTML.*

Esercizio P.6. *Scrivete un'applicazione Web che consente a un utente di specificare sei numeri della lotteria. Generate poi una combinazione vincente sul server e visua-lizzatele entrambe all'utente, insieme al conteggio dei numeri indovinati.*

Esercizio P.7. *Aggiungete il controllo degli errori all'esercizio precedente. Se i numeri della lotteria scelti dall'utente non rientrano nell'intervallo appropriato, oppure se ci sono duplicati, segnalatelo all'utente e consentite la correzione dell'errore.*

Esercizio P.8. *Scrivete un'applicazione che gestisca un carrello degli acquisti. Mostra-te all'utente un elenco di articoli che possono essere acquistati, uno dei quali può essere selezionato dall'utente (usate pulsanti radio). Quindi, chiedetene la quantità e visualizzate il nuovo contenuto del carrello. Infine, chiedete all'utente se vuole effettuare altri acquisti o se vuole pagare. Se vuole fare altri acquisti, tornate alla schermata iniziale, altrimenti chiedete nome e indirizzo dell'utente e visualizzate la fattura.*

Esercizio P.9. *Modificate l'applicazione del Paragrafo 5 che visualizza l'orario di più città, in modo che ciascuna città nell'elenco sia preceduta da una casella di spunta.*

Aggiungete un pulsante “Delete checked” che elimini tutte le città selezionate dall’utente.

Esercizio P.10. *Personalizzate l’applicazione che gestisce i fusi orari vista nel Paragrafo 3, in modo che l’utente inserisca dei dati per l’autenticazione (login) e specifichi una città nel suo profilo. Al successivo login di un utente, verrà visualizzato l’orario della città di sua scelta. Memorizzate in una base di dati gli utenti, le parole d’accesso e le città preferite. Serve un pulsante di disconnessione (logout) per cambiare utente.*

Esercizio P.11. *Estendete l’esercizio precedente in modo che un utente possa scegliere più città e che tutte le città scelte da un utente vengano memorizzate per essere visualizzate alla connessione successiva.*

Esercizio P.12. *Scrivete un servlet LotteryServlet che generi un insieme di numeri casuali estratti da una lotteria, come il JavaBean LotteryBean dei Consigli pratici 1.*

Document Outline

- *Multithreading*
  - *Obiettivi del capitolo*
  - *Nozioni fondamentali dei thread*
  - *Sincronizzazione*
  - *Evitare blocchi fatali*
  - *Un* *applicazione dei thread: Algoritmi animati*
  - *Riepilogo del capitolo*
  - *Classi, oggetti e metodi presentati nel capitolo*
  - *Esercizi di ripasso*
  - *Esercizi di programmazione*
- *Programmazione di rete*
  - *Obiettivi del capitolo*
  - *Il protocollo Internet*
  - *I protocolli al livello di applicazione*
  - *Un programma client*
  - *Un programma server*
  - *Connessioni URL*
  - *Inviare i dati di un modulo*
  - *Riepilogo del capitolo*
  - *Classi, oggetti e metodi presentati nel capitolo*
  - *Esercizi di ripasso*
  - *Esercizi di programmazione*
- *Connessione a base di dati*
  - *Obiettivi del capitolo*
  - *Organizzare le informazioni in una base di dati*
  - *Interrogazioni*
  - *Installare una base di dati*
  - *Programmazione di basi di dati in Java*
  - *Caso di studio: la base di dati di una banca*
  - *Concetti avanzati per le basi di dati*
  - *Riepilogo del capitolo*
  - *Ulteriori letture*

    - *Classi, oggetti e metodi presentati nel capitolo*
    - *Esercizi di ripasso*
    - *Esercizi di programmazione*
- *XML*
    - *Obiettivi del capitolo*
    - *Marcatori e documenti XML*
    - *Analisi sintattica di documenti XML*
    - *Creare documenti XML*
    - *Definizione di tipi di documento*
    - *Analisi sintattica con definizioni di tipi di documento*
    - *Riepilogo del capitolo*
    - *Ulteriori letture*
    - *Classi, oggetti e metodi presentati nel capitolo*
    - *Esercizi di ripasso*
    - *Esercizi di programmazione*
- *JavaServer Pages*
    - *Obiettivi del capitolo*
    - *Contenuto Web dinamico*
    - *Inserire le elaborazioni in oggetti JavaBean*
    - *Gestire i parametri della richiesta*
    - *Moduli (form) HTML*
    - *Gestione delle sessioni*
    - *Diramazioni e rimandi ad altre pagine*
    - *Un*[illegible]*applicazione a tre livelli*
    - *Servlet*
    - *Compilazione di pagine JSP*
    - *Riepilogo del capitolo*
    - *Ulteriori letture*
    - *Classi, oggetti e metodi presentati nel capitolo*
    - *Esercizi di ripasso*
    - *Esercizi di programmazione*